Application and evaluation of inductive reasoning methods for the semantic web and software analysis

Kiefer, Christoph ; Bernstein, Abraham

Abstract: Exploiting the complex structure of relational data enables to build better models by taking into account the additional information provided by the links between objects. We extend this idea to the Semantic Web by introducing our novel SPARQL-ML approach to perform data mining for Semantic Web data. Our approach is based on traditional SPARQL and statistical relational learning methods, such as Relational Probability Trees and Relational Bayesian Classifiers. We analyze our approach thoroughly conducting four sets of experiments on synthetic as well as real-world data sets. Our analytical results show that our ap- proach can be used for almost any Semantic Web data set to perform instance-based learning and classification. A comparison to kernel methods used in Support Vector Machines even shows that our approach is superior in terms of classification accuracy.

DOI: https://doi.org/10.1007/978-3-642-23032-5_10

Application and Evaluation of Inductive Reasoning Methods for the Semantic Web and Software Analysis

Christoph Kiefer and Abraham Bernstein * **

Dynamic and Distributed Information Systems Group
Department of Informatics, University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
http://www.ifi.uzh.ch/ddis
lastname@ifi.uzh.ch

Abstract. Exploiting the complex structure of relational data enables to build better models by taking into account the additional information provided by the links between objects. We extend this idea to the Semantic Web by introducing our novel SPARQL-ML approach to perform data mining for Semantic Web data. Our approach is based on traditional SPARQL and statistical relational learning methods, such as Relational Probability Trees and Relational Bayesian Classifiers. We analyze our approach thoroughly conducting four sets of experiments on synthetic as well as real-world data sets. Our analytical results show that our approach can be used for almost any Semantic Web data set to perform instance-based learning and classification. A comparison to kernel methods used in Support Vector Machines even shows that our approach is superior in terms of classification accuracy.

Keywords: Inductive Reasoning, Semantic Web, Machine Learning, SPARQL, Evaluation

Please Note: This paper represents part of the summer school lecture. It contains one critical, previously unpublished element: the description of inductive reasoning as an important component for non-traditonal reasoning on the Semantic Web. The lecture will also cover analogical reasoning [28,27,25], Markov Logic Networks [38], and the use of modern distributed techniques to run graph algorithms such as SIGNAL/COLLECT [43], Pregel [31], or MapReduce [12] with the Hadoop infrastructure (http://hadoop.apache.org/).
A. B.

* This paper is a significant extension and complete rewrite of [26], which won the best paper award at ESWC2008.

** This paper is published as: Christoph Kiefer and Abraham Bernstein, Application and Evaluation of Inductive Reasoning Methods for the Semantic Web and Software Analysis. *Reasoning Web* 2011: 460-503

1 Introduction

The vision of the Semantic Web is to interlink data from divers heterogeneous sources using a semantic layer as "glue" technology. The result of this combination process constitutes the often cited *Web of data* that makes data accessible on the traditional Web such that other applications can understand and reuse it more easily [5,1].

The above-mentioned semantic glue basically comprises a *rule-based meta-data layer* to *expose the meaning of data* in a machine-readable format. The term *rule-based* refers to the logic-based foundations of the Semantic Web that uses a number of description logic (DL) languages to represent the terminological knowledge of a domain (*i.e.*, a data source) in a structured and theoretically sound way. *Meta-data* means *self-describing*, that is, the raw data is tagged with additional information to express its meaning in the format of these DL languages.

The most universal DL languages in the Semantic Web are the *Resource Description Framework (RDF)*[1] and the *Web Ontology Language (OWL)*.[2] These languages/formats enable (i) to combine heterogeneous data under a common representation scheme by the use of *ontologies* and (ii) to give the data some well-defined, logic-based semantics, turning the otherwise meaningless data into information typically stored in a *knowledgebase (KB)*. Hence, ontologies serve as a formal specification of the conceptualization of this knowledge in terms of *classes* and *relations* among them [18].

1.1 Description Logic Reasoning

At this point, we are able to transfer the data that comes, for instance, from traditional relational databases to Semantic Web knowledgebases by using ontologies to specify the structure of the knowledge and a set of description logic languages to define the (logical) relations between these structure elements.

Typically, the information in a knowledgebase is stored as *asserted* (*i.e.*, *atomic*) facts. Such a piece of information could, for example, be the proposition "The type of service A is tourism", or in triples notation `[ serviceA type tourism ]`.

Now suppose the knowledgebase additionally includes the information `[ serviceB type serviceA ]` to express that service B is a specification of service A (B might, for instance, deliver information about hotels in a given city). One of the underpinnings of the Semantic Web and, therefore, a strength of any such semantic architecture is the ability to reason from the data, that is, to derive new knowledge (new facts) from base facts. In other words, the information that is already known and stored in the knowledgebase is extended with the information that can be *logically deduced from the ground truth*.

[1] `http://www.w3.org/RDF/`
[2] `http://www.w3.org/TR/owl-features/`

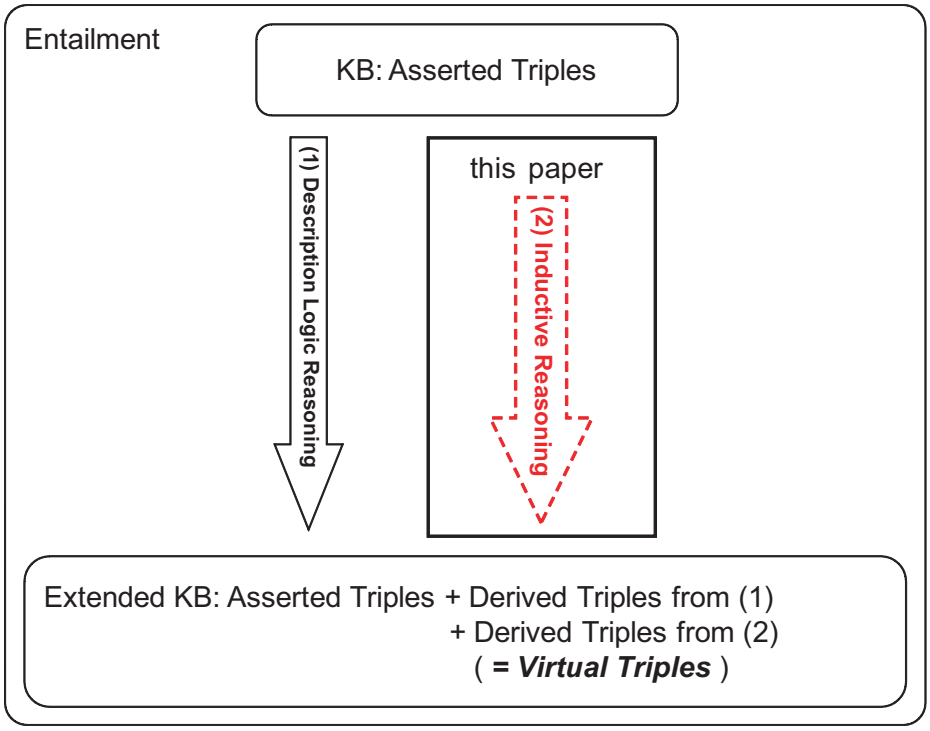

Fig. 1. The traditional Semantic Web infrastructure supports a logic-based access to the Semantic Web. It offers a retrieval (or reasoning) approach of data based on facts and classical deductive description logic reasoning (left arrow). Our novel reasoning extension presented and evaluated in this paper, on the other hand, extends the traditional Semantic Web infrastructure with *inductive reasoning* that is realized by *virtual triple patterns* and *statistical induction techniques* (right arrow).

This situation is also depicted in Figure 1 that shows schematically by the leftmost arrow the typical description logic reasoning process to infer additional, derived triples from a set of asserted triples in a knowledgebase. To summarize, the above service example is a simple application of *classical deductive logic* where the rule of inference over the type (subclass) hierarchy makes the proposition of B being of type tourism a valid conclusion.

1.2 What Is This Paper All About?

Metaphorically speaking, if this world would only be black and white, this is all that we could expect from a classical deductive reasoning system as supported by the current Semantic Web infrastructure. All the conclusions that could be drawn given some well-defined semantics such as the ones that come with the RDF/OWL languages will always be true if and only if the premises (*i.e.*, as-

serted knowledge, ground truth) are true. Otherwise they will be false, without any exception.

But the world is (fortunately!) not only black and white. The truth is, the world does generally not fit into a fixed, predetermined logic system of zeros and ones. Everyday life demonstrates again and again that we are performing some kind of *reasoning under uncertainty*, which does not follow the strict rules of formal logic.

Consider, for example, a doctor having to provide a medical diagnosis for one of his patients. Although he knows from his experiences and similar courses of disease that this special therapy seems to be best, there is, however, some risk involved, as such an inference is defeasible (*i.e.*, can be called into question)—medical advances may invalidate old conclusions. In other words, our actions are almost always driven by our heart and spirit (*i.e.*, by belief, experience, vague assumptions) rather than by formal logical implication.

To account for this, especially to deal with uncertainty inherent in the physical world, different models of human reasoning are required. Philosophers and logicians (among others) have, therefore, established new science fields in which they investigate and discuss such new types of human reasoning [32]. One prominent way to model human reasoning to some extend is *inductive reasoning* that denotes the process of *reasoning from sample-to-population* (*i.e.*, evidence-based reasoning). In inductive reasoning, the premises are only believed to support the conclusions but they cannot be (logically) entailed.

This paper transfers the idea of inductive reasoning to the Semantic Web and Software Analysis. To this end, it extends the well-known RDF query language SPARQL with our novel, non-deductive reasoning extension in order to enable inductive reasoning.

Traditional RDF query languages such as SPARQL [37] or SeRQL [9] support a logic-based access to the Semantic Web. They offer a retrieval approach of data based on facts and classical deductive description logic reasoning. The extension presented and evaluated in this paper, on the other hand, extends traditional Semantic Web query answering with inductive reasoning facilities.

Inductive reasoning is realized by *statistical induction techniques* which are applied to draw conclusions about an individual given some statistical quantities such as probabilities, averages, or deviations from a previous examined population. In other words, by the use of statistical induction techniques, additional triples are derived based on some (precomputed) statistics about these data.

Example 1 (Statistical Induction) *Suppose that from a set of 5 services, 3 are related to the tourism sector (*i.e.*, have type* `tourism`*) and 2 to the medical sector (see Figure 2). Given only this information, we could conclude that for a new, not yet examined service F (one that is outside the original sample of five services), there is a probability estimate of 0.6 (*i.e.*, $\frac{3}{5}$) that the service is of type* `tourism`*. Because the probability estimate for type* `medical` *is only 0.4 (*i.e.*, $\frac{2}{5}$), we infer that F must also be located in the tourism sector. Such inferences are also called* quantitative probabilistic reasoning *[32].*

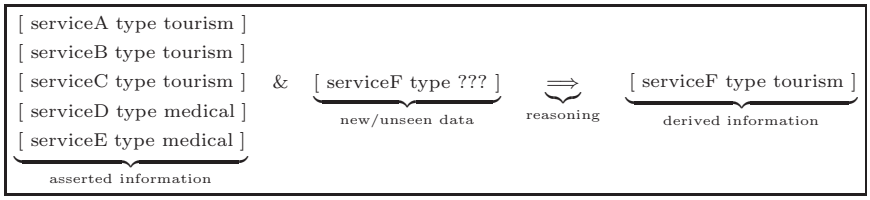

Fig. 2. Novel inductive reasoning process using statistical inference methods.

The inductive reasoning approach presented in this paper works similarly: it involves a prediction/classification step performed by the SPARQL query engine to predict, for instance, the membership of a data sample (individual/instance) to a particular class with some prediction accuracy. For the classification task, this approach employs algorithms from machine learning such as decision trees, support vector machines (SVMs), and regression models [45].

1.3 Our Approach

To address these issues, specifically to implement our novel reasoning variant by using SPARQL, this paper introduces the concept of *virtual triple patterns (VTPs)*. Figure 1 shows the relation between asserted, 'ordinary' derived (1), and extraordinary derived triples (2). Ordinary triples are inferred using the traditional description logic reasoning system of the Semantic Web by applying the fundamental RDF/OWL inference rules. The extraordinary triples are the result of applying our novel inductive reasoning methods to the Semantic Web.

Typically, a Semantic Web dataset is made of a large number of RDF triples which model the relations among all data instances in terms of a so called `subject` and `object`, and a `predicate` to link them up. As an example, consider the triple pattern `[ serviceA hasName name ]` that relates service A to its name by the `hasName` predicate. An RDF dataset can then be thought of as a graph which is spanned by these triples. Query evaluation, can thus, essentially be reduced to the task of matching a number of triple patterns (called graph patterns) to an RDF graph.

VTPs, on the other hand, are triple patterns which are *not* matched against an RDF graph. Instead, they perform pattern matching as the result of calling some user-defined piece of code. VTPs can conceptually be thought of as ordinary function calls which consist of the function name followed by a list of arguments in parentheses, and which have a return value. VTPs are presented in details in Section 3.

1.4 Importance to the Semantic Web and Software Analysis

Regarding inductive reasoning, a number of past researches have highlighted the crucial element of statistics for the Semantic Web (*e.g.*, [17] or [21]). Two promi-

nent tasks that can benefit from the use of statistics are *Semantic Web service classification* and *(semi-) automatic semantic data annotation*. Therefore, the support from tools that are able to work autonomously is needed to add the required semantic annotations. Consequently, a big challenge for Semantic Web research is not if, but *how* to extend the existing Semantic Web infrastructure with statistical inferencing capabilities.

In Software Analysis, researchers heavily deal with the analysis of software source code and abstract software models. Software Analysis and its subdisciplines have grown tremendously, which can also be observed from the increasing number of diverse papers submitted to the largest Software Analysis/Engineering conferences and workshops such *ICSE*[3] and *MSR*[4] in the past years. In order to show the advantages of inductive reasoning for Software Analysis via virtual triple patterns and Statistical Relational Learning methods, we have decided to perform a *bug prediction experiment* where the goal is to predict whether or not a piece of code is likely to have bugs or not. Roughly speaking, software bug prediction (aka defect prediction) is about finding locations in source code that are likely to be error-prone. We, thus, argue that the development and testing of tools that are able to detect such defect locations are crucial to (i) increase software quality and (ii) to reduce software development cost (among others).

To summarize, as we will show in this paper, the Semantic Web and Software Analysis can substantially benefit from our novel inductive reasoning extension to SPARQL. Our proposed, unified, SPARQL-based framework not only helps to solve these important research tasks, but also helps to establish the semantic glue mentioned at the very beginning of this work by (semi-) automatic semantic annotation (through classification).

Specifically, the contributions can be summarized as follows: For our inductive reasoning extension, we first present our *SPARQL-ML* approach to create and work with statistical induction/data mining models in traditional SPARQL (see Section 3). The major contribution of our proposed SPARQL-ML framework is, therefore, the ability *to support data mining tasks for knowledge discovery in the Semantic Web* .

Second, our presented SPARQL-ML framework is validated using not less than four case studies ranging over three heavily researched Semantic Web tasks and one Software Analysis task. For the Semantic Web, we perform two general data classification tasks (Sections 4.1 and 4.2) and one specific semantic service classification task (*i.e.*, service annotation; see Section 4.3). For Software Analysis, we perform a bug prediction task using semantically annotated software source code (Section 4.4).

By applying our approaches to these different tasks, we hope to show the approach's generality, ease-of-use, extendability, and high degree of flexibility in terms of customization to the actual task. Finally, we close the paper with a

[3] International Conference on Software Engineering, `http://www.icse-conferences.org/`

[4] International Working Conference on Mining Software Repositories, `http://msr.uwaterloo.ca/`

discussion of the results in Section 5, and our conclusions and some insights into future work in Section 6.

2 Related Work

This chapter briefly reviews the most important related work. We start with a short summary of some important Semantic Web publications to set this work into perspective in Section 2.1. Specifically, we review a couple of studies that influenced the history and development of SPARQL. Section 2.2 proceeds with some related approaches to inductive reasoning. Section 2.3 proceeds with some of the most important related works regarding the tasks we use to validate/evaluate our SPARQL-ML framework.

2.1 Semantic Web

In 1989, Alexander Borgida [8] presented his work about the CLASSIC language that can be regarded as an early approach to the Semantic Web. CLASSIC is a language for structural, partial descriptions of objects in a relational database management system. It is worth to mention this work for several reasons: first, CLASSIC allows the user to describe both the intensional structure of objects as well as their extensional relations to other objects (which in RDF terminology is achieved through data and object type properties); second, using CLASSIC it is possible to describe objects only partially and to add more information about it over time; third, CLASSIC can be used both as a data description as well as data query language; and fourth, the CLASSIC system is able to infer new knowledge about objects (*i.e.*, it performs an early kind of reasoning by applying a limited form of forward-chaining rules [39]).

12 years later, in 2001, Tim Berners-Lee [2] published his famous article about his vision of a true Semantic Web as an extension of the current Web in which data is given well-defined meaning through *ontologies*. This is an important improvement to, for instance, XML that allows the user to structure the data but does not say what the data in fact means. Such semantically enriched data can then be meaningfully manipulated by autonomous computer programs also referred to as *agents*.

Furthermore, one of the most important building blocks of the Semantic Web are, as argued in [2], automated reasoning facilities, which denote the process of deriving new information from existing, asserted information through classical deductive description logic (DL) reasoning rules. Pure deductive DL reasoning is, however, not sufficient for some tasks. On the contrary, as we will show in this work, tasks such as semantic service classification can substantially benefit from our novel, inductive reasoning facility.

Five years later, Shadbolt, Hall, and Berners-Lee [41] critically revisited some of the statements made in [2]. Specifically, they emphasized on the need for shared semantics which is badly needed for data integration—a task that is of particular importance in the life sciences [30]. As explained in [41], most of the

motivation for a Semantic Web came from the tremendous amount of valuable information stored in traditional relational databases. This information must be exported into a system of URIs and, hence, given well-defined meaning. "The data exposure revolution has, however, not yet happened", which should increase the amount of available RDF data to push the Semantic Web even further.

RDF Query Language SPARQL In recent years, the RDF query language SPARQL has gained increasing popularity in the Semantic Web. SPARQL stands for *SPARQL Protocol and RDF Query Language* and offers well-known constructs from database technology, such as `SELECT`, `FILTER`, and `ORDER BY`. Furthermore, the SPARQL specifications define a protocol for the communication between a query issuer and a query processor. The SPARQL language has currently the status of a W3C Recommendation and is extensively described in [37].

As the language was used more and more over time by different parties for different applications, it became clear that it needed a more mathematical basis in terms of an algebra, similar to relational algebra for relational databases [10]. This was especially important as the need for optimization of SPARQL queries also arose as people wanted to use ever growing RDF datasets for their experiments. Among those who dealt with the development of an algebra for SPARQL, it was Cyganiak [11] who described as one of the first how to transform (a subset of) SPARQL into relational algebra that is, as argued by Cyganiak, the language of choice when analyzing queries in terms of query planning and optimization. Furthermore, he defined the semantics of the relational algebra operators and discussed a translation into SQL, which is important to execute the queries against traditional relational databases storing the RDF data.

One year after Cyganiak's work was published, Pérez [35] conducted an extensive analysis of the semantics and complexity of SPARQL, focusing on the algebraic operators `JOIN`, `UNION`, `OPTIONAL`, and `FILTER`. The semantics and complexity of these operators are studied in great detail and insights into query optimization possibilities are presented. In particular, they introduced well-defined graph patterns that can be transformed to patterns in normal form, which when matched against the underlying RDF dataset results in improved query execution time. The presented theoretical framework in [35] is build around sets of solution mappings which are created in the process of matching the query's basic graph patterns (BGP) to the underlying RDF graph.

It is important to say, that the study of Pérez *et al.* highly influenced the work presented in this paper. Our proposed inductive reasoning extension to SPARQL is based on *virtual triple patterns* (see Section 3.2) that are theoretically defined in the algebraic notation of [35]. *ARQ property functions*[5]—the implementational foundations of virtual triple patterns—are, however, not addressed in [35]. It is, therefore, one of the contributions of this work to reflect on the semantics of such property functions, as our SPARQL-ML framework heavily relies on them.

[5] `http://jena.sourceforge.net/ARQ/library-propfunc.html`

2.2 Inductive Reasoning

Our proposed inductive reasoning extension relies on statistics (*i.e.*, machine learning techniques) and elements from probability theory to reason from data. In this section, we will briefly review some of the inductive reasoning (machine learning) approaches from the Semantic Web literature which are relevant in the context of this work. Specifically, as our novel reasoning extension heavily relies on *Statistical Relational Learning (SRL)* algorithms, we shortly summarizes the two SRL methods we use in this paper. The section closes with an overview of some related works regarding the Semantic Web and Software Analysis tasks we chose to evaluate our inductive reasoning extension.

Little work has been done so far on seamlessly integrating knowledge discovery capabilities into SPARQL. Recently, Kochut and Janik [29] presented *SPARQLeR*, an extension of SPARQL to perform semantic association discovery in RDF (*i.e.*, finding complex relations between resources). One of the main benefits of our inductive reasoning approach through SPARQL-ML is that we are able to use a multitude of different, pluggable machine learning techniques to not only perform semantic association discovery, but also prediction/classification and clustering.

Getoor and Licamele [16] highlighted the importance of link mining for the Semantic Web. They state that the links between resources form graphical patterns which are helpful for many data mining task, but usually hard to capture with traditional statistical learning approaches. With our SPARQL-ML framework we, therefore, apply SRL algorithms that are able to exploit these patterns to improve the performance of the pure statistical approaches (see Section 3.2).

Similarily, Gilardoni [17] argued that machine learning techniques are needed to build a semantic layer on top of the traditional Web. Therefore, the support from tools that are able to work autonomously is needed to add the required semantic annotations. We show that our inductive reasoning extension to SPARQL offers this support, and thus, facilitates the process of (semi-) automatic semantic annotation (through classification).

We are aware of two other independent studies that focus on data mining techniques for Semantic Web data using Progol—an Inductive Logic Programming (ILP) system.[6] In the first study, Edwards [14] conducted an empirical investigation of the quality of various machine learning methods for RDF data classification, whereas in the second study, Hartmann [19] proposed the *ARTEMIS* system that provides data mining techniques to discover common patterns or properties in a given RDF dataset. Our work extends their suggestions in extending the Semantic Web infrastructure in general with machine learning approaches, enabling the exploration of the suitability of a large range of machine learning techniques (as opposed to few ILP methods) to Semantic Web tasks without the tedious rewriting of RDF datasets into logic programming formalisms.

Last but not least, Bloehdorn and Sure [6] explored an approach to classify ontological instances and properties using SVMs (*i.e.*, kernel methods). They

[6] `http://www.doc.ic.ac.uk/~shm/progol.html`

presented a framework for designing such kernels that exploit the knowledge represented by the underlying ontologies. Inspired by their results, we conducted the same experiments using our proposed SPARQL-ML approach (see Section 4.3). Initial results show that we can outperform their results by a factor of about **10%**.

Statistical Relational Learning Methods Our SPARQL-ML framework employs machine learning-based, statistical relational reasoning techniques to create and work with data mining models in SPARQL (see Section 3). These techniques are *Relational Probability Trees (RPTs)* and *Relational Bayesian Classifiers (RBCs)* that model not only the intrinsic attributes of objects, but also the extrinsic relations to other objects and, thus, should perform at least as accurate as traditional, propositional learning techniques. Both algorithms enable to perform inductive reasoning for the Semantic Web, in other words, they enable to induce statistical models *without prior propositionalization of the data* (*i.e.*, translation to a single table) [13], which is a cumbersome and error-prone task.

RPTs [33] extend standard probability estimation trees (also called *decision trees*) to a relational setting, in which data instances are heterogeneous and interdependent. This procedure is explained in more details in Section 3.2.

The RBCs used to perform inductive reasoning through SPARQL-ML were also proposed by Neville in [34]. An RBC is a modification of the traditional Simple Bayesian Classifier (SBC) for relational data [45]. Please refer to Section 3.2 for more details about RBCs.

2.3 SPARQL-ML Evaluation/Validation Tasks

Sabou [40] stated that the Semantic Web can facilitate the discovery and integration of web services. The addition of ontologies, containing knowledge in the domain of the service such as the types of input/output parameters, offers new background information, which can be exploited by machine learning algorithms. We evaluate this assumption in this work in the context of our semantic web service classification experiment by comparing the results of data mining with and without the enhancement of ontologies (see Section 4.2).

Furthermore related is the study of Heß [21], in which a machine learning approach for semi-automatic classification of web services is described. Their proposed application is able to determine the category of a WSDL web service and to recommend it to the user for further annotation. They treated the determination of a web service's category as a text classification problem and applied traditional data mining algorithms, such as Naïve Bayes and Support Vector Machines [45]. Our conducted experiment is similar in that it employs OWL-S service descriptions instead of WSDL descriptions. In contrast to [21], we employ SRL algorithms such as RPTs and RBCs and additional background information provided by ontologies to perform semantic service classification. Regarding bug/defect prediction in source code, many approaches have been proposed in the past to accomplish this task. In Fenton [15], an extensive survey

and critical review of the most promising learning algorithms for bug prediction from the literature is presented. [15] proposed to use Bayesian Belief Networks (BBNs) to overcome some of the many limitations of the reviewed bug prediction algorithms. BBNs are based on applying Bayes' rule that assumes that all attributes of training and testing examples are independent of each other given the value of the class variable (which is called conditional independence). It is important to note that the RBCs validated in this case study is an extension of the simple Bayesian classifier (that applies Bayes's rule for classification) to a relational data setting (see Section 3.2).

Bernstein [3] proposed an approach based on a non-linear model on temporal features for predicting the number and location of bugs in source code. In their experiments, six different models were trained using Weka's J48 decision tree learner. The data they used to evaluate their prediction models were collected from six plug-ins of the Eclipse open source project.[7]

These data were then enhanced with temporal information extracted from Eclipse's concurrent versions system (CVS) and information from Bugzilla.[8] Using this approach, they successfully showed that the use of a non-linear model in combination with a set of temporal features is able to predict the number and location of bugs with a very high accuracy.

In order to demonstrate the usefulness and applicability of inductive reasoning on semantically annotated software source code, we perform the same experiment using our proposed SPARQL-ML framework (see Section 4.4). As we will show in the remainder of this paper, inductive reasoning techniques for this kind of task and dataset provide a powerful means to quickly analyze source code.

3 Inductive Reasoning with SPARQL-ML

This chapter presents our novel inductive reasoning approach that intends to complement the classical deductive description logic reasoning facilities of the traditional Semantic Web. In a nutshell, inductive reasoning enables to draw conclusions about an unseen object (not included in the original set of observed samples) based on statistical induction/inferencing techniques. Basically, this comprises (1) the learning of a statistical model mirroring the characteristics of the observed samples and (2) the application of the model to the population. In Semantic Web terminology, inductive reasoning denotes the process of deriving new triples from the set of asserted triples based on the statistical observations of a sufficiently large, representative set of resources.

To add inductive reasoning support to the current Semantic Web infrastructure, specifically to integrate it with SPARQL, we focus on a special class of statistical induction techniques called *statistical relational learning (SRL)* methods. As we will show in our experiments, the large and continuously growing amount of interlinked Semantic Web data is a perfect match for SRL methods

[7] `http://www.eclipse.org/`
[8] `http://www.bugzilla.org/`

due to their focus on *relations between objects* in addition to features/attributes of objects of traditional, propositional learning techniques.

Our inductive reasoning extension to SPARQL is called *SPARQL-ML (SPARQL Machine Learning)*. SPARQL-ML supports the integration of traditional Semantic Web techniques and machine learning-based, statistical inferencing to create and work with data mining models in SPARQL. To that end, SPARQL-ML introduces new keywords to the official SPARQL syntax to facilitate the induction of models.

For the prediction/classification of unseen objects in a dataset, SPARQL-ML makes use of our proposed *virtual triple pattern* approach [27] to call customized, external prediction functions implemented as ARQ property functions (Section 3.2).

The two SRL methods used in SPARQL-ML are *Relational Probability Trees (RPTs)* and *Relational Bayesian Classifiers (RBCs)* proposed in [33] and [34], respectively. The use of these methods enables to induce statistical models without prior propositionalization (*i.e.*, translation to a single table) [13]—a cumbersome and error-prone task.

To ensure the extensibility of our inductive reasoning approach with other learning methods, the *SPARQL Mining Ontology (SMO)* is proposed to enable the seamless integration of additional machine learning techniques (see Section 3.3).

3.1 Preliminaries

In this chapter, the dataset D shown in Figure 3 will be used for all examples. D describes three semantic services A, B, and C in triple notation (with profile names `SP1`, `SP2`, and `SP3` respectively). In triple notation, each characteristic of the services is written as a simple triple of *subject*, *predicate*, and *object*, in that order. Note that all the queries in the remainder of this chapter use the prefixes shown in Listing 1.1.

3.2 Theoretical Foundations

The theory introduced in this chapter heavily relies on our virtual triple pattern approach presented in [27] and Statistical Relational Learning learning methods. This section, therefore, (i) briefly reviews the most important elements of the semantics of SPARQL and virtual triples, and (ii), shortly summarizes Relational Bayesian Classifiers (RBCs) and Relational Probability Trees (RPTs).

Semantics of SPARQL To explain our virtual triple pattern approach, the concept of SPARQL solution mappings is central. According to [37], a solution mapping is defined as follows:

Definition 1 (Solution Mapping) *A solution mapping* $\mu(?v \mapsto t)$ *maps a query variable* $?v \in V$ *to an RDF term* t *where* V *is the infinite set of query*

```
D = {
 (SP1 profile:name ‘‘CityLuxuryHotelInfoService’’),
 (SP1 profile:desc ‘‘Often used service to get
                   information about luxury hotels.’’),
 (SP1 profile:hasInput _CITY),
 (SP1 profile:hasInput _COUNTRY),
 (SP1 profile:hasOutput _LUXURYHOTEL),
 (SP1 profile:hasCategory ‘‘travel’’),
 (SP2 profile:name ‘‘CityCountryHotelInfoService’’),
 (SP2 profile:desc ‘‘Accommodation and restaurant
                   information service.’’),
 (SP2 profile:hasInput _CITY),
 (SP2 profile:hasOutput _HOTEL),
 (SP2 profile:hasCategory ‘‘travel’’),
 (SP3 profile:name ‘‘CityCountryInfoService’’),
 (SP3 profile:desc ‘‘Hotels and sports facilities
                   information service.’’),
 (SP3 profile:hasInput _SPORT),
 (SP3 profile:hasOutput _CAPITAL),
 (SP3 profile:hasCategory ‘‘education’’) }
```

Fig. 3. Example dataset D that lists services A, B, and C in triple notation.

variables and t a member of the set union of literals, IRIs, and blank nodes called RDF-T. The domain of μ, $dom(\mu)$, is the subset of V where μ is defined.

Example 2 (Solution Mappings) *Matching the basic graph pattern* { `SP1 profile:name ?name` } *against dataset D will result in a simple solution mapping,* i.e.*,*

$$\mu(?name \mapsto \text{“CityLuxuryHotelInfoService”}).$$

The domain of μ is $dom(\mu) =$ { *?name* } *(*i.e.*, μ is defined for precisely one variable). Matching the graph pattern* { `SP1 ?predicate ?name` } *against D will additionally find a mapping for variable* `?predicate`*,* i.e.*,*

$$\mu(?predicate \mapsto profile{:}name,$$
$$?name \mapsto \text{“CityLuxuryHotelInfoService”}).$$

In this case, the domain of μ is $dom(\mu) =$ { *?predicate*, *?name* }*.*

In [35], it is stated that the evaluation of a graph pattern over a dataset results in a *(multi-) set of solution mappings Ω.*

Example 3 (Set of Solution Mappings) *The basic graph pattern* { `?profile profile:name ?name` } *specifies both the subject and the object of the triple pattern as variable. The graph matching algorithm will return a set of solution mappings Ω including precisely three solution mappings when*

```
PREFIX pf: <java:ch.uzh.ifi.ddis.pf>
PREFIX grounding:
 <http://www.daml.org/services/owl-s/1.1/Grounding.owl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX process:
 <http://www.daml.org/services/owl-s/1.1/Process.owl#>
PREFIX profile:
 <http://www.daml.org/services/owl-s/1.1/Profile.owl#>
PREFIX rdf:
 <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX service:
 <http://www.daml.org/services/owl-s/1.1/Service.owl#>
PREFIX sml: <java:ch.uzh.ifi.ddis.pf.sml>
PREFIX smo: <http://www.ifi.uzh.ch/ddis/sparql-ml/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

Listing 1.1. Query prefixes used in this paper.

matching the pattern against dataset D, i.e.*,*

$$\begin{aligned}\Omega = \{\ & \mu_1(?profile \mapsto \textit{SP1}, \\ & \quad ?name \mapsto \textit{“CityLuxuryHotelInfoService”}), \\ & \mu_2(?profile \mapsto \textit{SP2}, \\ & \quad ?name \mapsto \textit{“CityCountryHotelInfoService”}), \\ & \mu_3(?profile \mapsto \textit{SP3}, \\ & \quad ?name \mapsto \textit{“CityCountryInfoService”})\ \}.\end{aligned}$$

Virtual Triple Pattern Approach Our proposed approach to enable inductive reasoning via SPARQL exploits ARQ property functions (aka *magic properties*).[9] The concept behind property functions is simple: whenever the predicate of a triple pattern is prefixed with a special name, a call to a customized, external prediction function (CPF) is made and arguments are passed to the function (in this case by the object of the triple pattern). The passed object may be an arbitrary list of query variables for which solution mappings were already found during query execution. The property function determined by the property URI computes a value and returns it to the subject variable of the triple pattern.

We call this the *virtual triple pattern approach* as such triple pattern expressions including property functions are not matched against the underlying ontology graph, but against the only virtually existing class membership of the resource specified in the pattern expression. More formally, a virtual triple pattern expression *vt* is defined as a triple employing a particular kind of property function reference by a property URI:

Definition 2 (Virtual Triple Pattern) *A virtual triple pattern vt is a triple of the form* { `?v pf:funct ArgList` } *where* `pf:funct` *is a property function and* `ArgList` *a list of solution mapping arguments* $\mu(?x_1 \mapsto t_1), \mu(?x_2 \mapsto$

[9] `http://jena.sourceforge.net/ARQ/extension.html#propertyFunctions`

```
1 SELECT ?descLower WHERE
2   { SP1       profile:desc   ?desc .
3     ?descLower  pf:lower-case  ( ?desc ) .
4   }
```

Listing 1.2. SPARQL query with a single virtual triple pattern expression including property function `lower-case` to convert the text argument to lower case.

$t_2), \ldots, \mu(?x_n \mapsto t_n)$. *The value computed by* `pf:funct` *is bound to the subject variable* `?v`.

Similarly to the definition of solution mappings, *virtual solution mappings* can now be defined.

Definition 3 (Virtual Solution Mapping) *A virtual solution mapping* $\mu_v(?v \mapsto t)$ *maps a query variable* $?v \in V$ *to an RDF term* t *where* V *is the infinite set of query variables and* t *an RDF literal* ***not included*** *in the queried RDF graph. The domain of* μ_v, $dom(\mu_v)$, *is the subset of* V *where* μ_v *is defined.*

The sets of virtual solution mappings μ_v are defined as Ω_{VGP} and the sets of solution mappings found by basic graph pattern matching as Ω_{BGP}. Furthermore, based on the description of basic graph patterns in [37], *virtual graph patterns* VP are defined as sets of virtual triple patterns vt.

Example 4 (Virtual Solution Mapping) *Consider the query shown in Listing 1.2. Matching the first triple pattern on line 2 against dataset D results in the following set of solution mappings:*

$$\Omega_{BGP} = \{\ \mu(?desc \mapsto \text{“Often used service to get information about luxury hotels.”})\ \}.$$

The evaluation of the virtual triple pattern on line 3 results in a call to the property function `lower-case`*, which results in the set* Ω_{VGP} *of a single virtual solution mapping,* i.e.*,*

$$\Omega_{VGP} = \{\ \mu_v(?descLower \mapsto \text{“often used service to get information about luxury hotels.”})\ \}.$$

Statistical Relational Learning (SRL) Methods SRL methods have been shown to be very powerful as they model not only the *intrinsic attributes* of objects, but also the *extrinsic relations* to other objects, thus, should perform at least as accurate as traditional, propositional learning techniques (cf. [13], [33], and [34]).

Note that in accordance with [33], we refer to such objects with links to intrinsic and extrinsic attributes as *subgraphs*: “The SRL algorithms take a collection of subgraphs as input. Each subgraph contains a single target object to

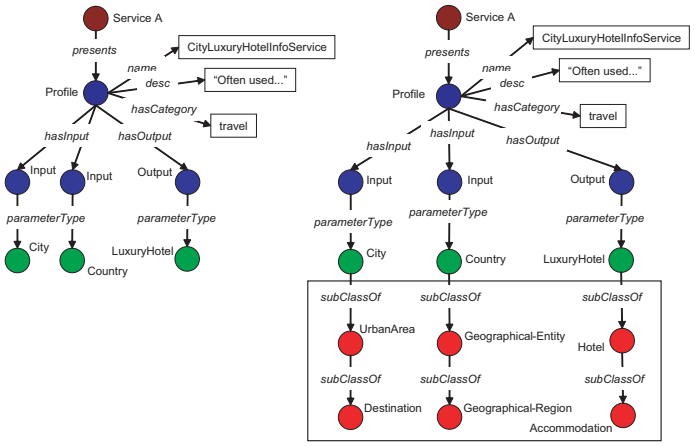

Fig. 4. Intrinsic vs. extrinsic attributes of the semantic service A. The subgraph on the left contains A's intrinsic relations to its attributes, whereas on the right the extrinsic relations are shown. These extrinsic relations are the `subClassOf` links to the super concepts of A's asserted (*i.e.*, direct) I/O concepts.

be classified; The objects and links in the subgraph form its relational neighborhood."

Example 5 (Intrinsic vs. Extrinsic Attributes) *Consider Figure 4 that shows service A represented as relational subgraph. The subgraph on the left contains the service profile of A, the links to its name, description, and category, as well as the links to its in- and output concepts. These objects and links in the subgraph are called* intrinsic *as they are* directly associated *with A.*

The subgraph on the right basically models the same information about A but is extended with extrinsic *relations to other objects. In this example, these relations are the* `subClassOf` *links to the super concepts of A's asserted (i.e., direct) I/O concepts. Of course, these relations could again have other relations to other objects resulting in an even larger relational neighborhood of A.*

Relational Bayesian Classifiers (RBCs). An RBC is a modification of the traditional Simple Bayesian Classifier (SBC) for relational data [34] (also called *Naïve Bayes Classifier*). SBCs assume that the attributes of an instance C are conditionally independent of each other given the class of the instance. Hence, the probability of the class given an example instance can be computed as the product of the probabilities of the example's attributes $A_1, \ldots, A_n$ given the

Service	Category	Inputs	Input Super Concepts (ISCs)
A	travel	{ travel.owl#City, portal.owl#Country }	{ Destination, Generic-Agent, Geographical-Entity, Geographical-Region, Location, support.owl#Thing, Tangible-Thing, Temporal-Thing, UrbanArea }
B	travel	{ portal.owl#City, portal.owl#Country }	{ Generic-Agent, Geographical--Entity, Geographical-Region, Location, Municipal-Unit, support.owl#Thing, Tangible-Thing, Temporal-Thing }
C	education	{ Sports }	{ Activity }

Table 1. The relational subgraphs of the semantic services A, B, and C are decomposed by attributes. The table lies the focus on the input concepts. Each column represents one of the service's attributes and the cells contain the multisets (or distributions) of values of these attributes.

Service	Category	Outputs	Output Super Concepts (OSCs)
A	travel	{ LuxuryHotel }	{ Accommodation, Hotel }
B	travel	{ Hotel }	{ Accommodation }
C	education	{ Capital }	{ Destination, travel.owl#City, UrbanArea }

Table 2. The relational subgraphs of the semantic services A, B, and C are decomposed by attributes (focus on output concepts).

class, *i.e.*,

$$\begin{aligned} Pr(C = c_i \,|\, A_1, \ldots, A_n) \\ &= \alpha Pr(A_1, \ldots, A_n \,|\, C = c_i) Pr(C = c_i) \\ &= \alpha Pr(C = c_i) \times \prod_{i=1}^{n} Pr(A_i \,|\, C = c_i). \end{aligned} \tag{1}$$

Equation 1 is exactly Bayes' rule of conditional probability where α is a scaling factor dependent only on the attributes $A_1, \ldots, A_n$.

RBCs apply this independence assumption to relational data. The RBC algorithm transforms the heterogeneous subgraphs in Figure 4 to homogenous sets of attributes as shown in Tables 1 and 2. Each row in the tables stands for a subgraph (*i.e.*, semantic service), each column represents one of its attributes, and the cells contain the multisets (or distributions) of values of attributes. These attributes include the service category as well as the asserted and inferred I/O concept distributions of the semantic services.

Learning an RBC model then basically consists of estimating probabilities for each attribute and/or attribute-value distribution. Such probability estimation techniques include, but are not limited to, *average-value* and *random-value* estimations (cf. [34]).

Relational Probability Trees (RPTs). RPTs extend standard probability estimation trees (also called *decision trees*) to a relational setting, in which data

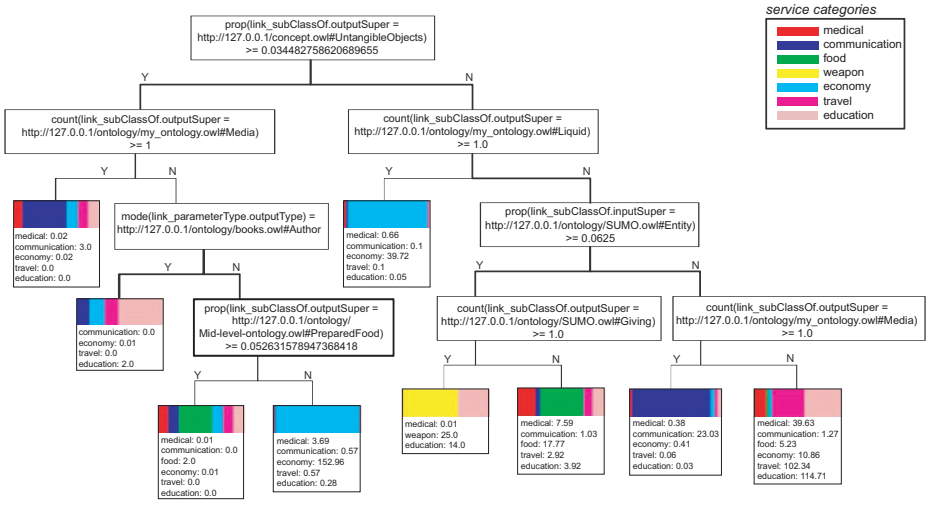

Fig. 5. Example RPT to predict the value for a semantic service's `hasCategory` attribute.

instances are heterogeneous and interdependent [33].[10] Similar to RBCs, RPTs look beyond the intrinsic attributes of objects, for which a prediction should be made; it also considers the effects of adjacent objects (extrinsic relations) on the prediction task.

As is the case for RBCs, the RPT algorithm first transforms the relational data (the semantic services represented as subgraphs) to multisets of attributes. It then attempts to construct an RPT by searching over the space of possible binary splits of the data based on the relational features, until further processing no longer changes the class distributions significantly. The features for splitting these (training) data are created by mapping the multisets of values into single-value summaries with the help of aggregation functions. These functions are for instance *count*, *mode/average*, *degree*, *proportion*, *minimum*, *maximum*, and *exists* (see [33]).

Example 6 (RPT Classification) *As an example, consider the RPT shown in Figure 5 that predicts the value for a semantic service's* `hasCategory` *attribute. The value of this attribute should be one out of* `communication`*,* `economy`*,* `education`*,* `food`*,* `medical`*,* `travel`*, and* `weapon`*. The root node in the RPT starts by examining the super concepts of the service's direct (i.e., asserted) output concepts in the current subgraph. If the proportion of all super*

[10] Actually, when it comes to predicting numeric values, decision trees with averaged numeric values at the leaf nodes are called *regression trees*.

concepts being `concept.owl#UntangibleObjects` *is greater than or equal to 0.0345, the left edge in the RPT is traversed. Assume no, the next test looks at how many super concepts have type* `my_ontology.owl#Liquid` *in the subgraph, represented by* `count(link_subClassOf.outputSuper = my_ontology.owl#Liquid)`*. Specifically, we test whether the subgraph contains at least one such super concept. If this is the case, we pass this test and traverse the left edge to the leaf node.*

The leaf nodes show the distribution of the training examples (that "reached the leaf") and the resulting class probabilities of the `hasCategory` *target attribute. In other words, the leaf nodes hold the probabilistic counts (out of all services from the training set that reach this leaf node) for each potential classification of this service. We can observe that services that reach this leaf node have much more likely category* `economy` *than any other category. Therefore, this model would predict that this service (subgraph) has category* `economy`*.*

3.3 Adding Inductive Reasoning Support to SPARQL Via SRL Methods

SPARQL-ML is an extension of SPARQL that extends the Semantic Web query language with knowledge discovery capabilities. Our inductive reasoning extensions add new syntax elements and semantics to the official SPARQL grammar described in [37]. In a nutshell, SPARQL-ML facilitates the following two tasks on any Semantic Web dataset: (1) induce a model based on training data using the new `CREATE MINING MODEL` statement (Section 3.3); and (2), apply a model to make predictions via two new ARQ property functions (Section 3.3). The model created in the `CREATE MINING MODEL` step follows the definitions in our *SPARQL Mining Ontology (SMO)* presented in Section 3.3.

SPARQL-ML is implemented as an extension to ARQ—the SPARQL query engine for Jena.[11] The current version of SPARQL-ML supports, but is not limited to *Proximity*[12] and *Weka*[13] as data mining modules.

Step 1: Learning a Model Syntax and Grammar. SPARQL-ML enables to induce a classifier (model) on any Semantic Web training data using the new `CREATE MINING MODEL` statement. The chosen syntax was inspired by the Microsoft Data Mining Extension (DMX) that is an extension of SQL to create and work with data mining models in Microsoft SQL Server Analysis Services (SSAS) 2005.[14] The extended SPARQL grammar is tabulated in Table 3. Listing 1.3 shows a particular example query to induce an RPT model for the prediction of the category of a semantic service.

[11] `http://jena.sourceforge.net/`
[12] `http://kdl.cs.umass.edu/proximity/index.html`
[13] `http://www.cs.waikato.ac.nz/ml/weka/`
[14] `http://technet.microsoft.com/en-us/library/ms132058.aspx`

[1]	*Query*	::=	Prologue(SelectQuery | ConstructQuery | DescribeQuery | AskQuery | CreateQuery)
[100]	*CreateQuery*	::=	CREATE MINING MODEL' SourceSelector '{' Var 'RESOURCE' 'TARGET' (Var ('RESOURCE' | 'DISCRETE' | 'CONTINUOUS') 'PREDICT'?)+ '}' DatasetClause* WhereClause SolutionModifier UsingClause
[102]	*UsingClause*	::=	'USING' SourceSelector BracketedExpression

Table 3. Extended SPARQL grammar for the `CREATE MINING MODEL` statement.

Our approach adds the `CreateQuery` symbol to the official SPARQL grammar rule of *Query* [37]. The structure of `CreateQuery` resembles the one of `SelectQuery`, but has complete different semantics: the `CreateQuery` expands to Rule 100 adding the new keywords `CREATE MINING MODEL` to the grammar followed by a `SourceSelector` to define the name of the trained model. In the body of `CreateQuery`, the variables (attributes) to train the model are listed. Each variable is specified with its content type, which is currently one of the following: `RESOURCE`—variable holds an RDF resource (IRI or blank node), `DISCRETE`—variable holds a discrete/nominal literal value, `CONTINUOUS`—variable holds a continuous literal value, and `PREDICT`—tells the learning algorithm that this feature should be predicted. The first attribute is additionally specified with the `TARGET` keyword to denote the resource for which a feature should be predicted (also see [33]).

After the usual `DatasetClause`, `WhereClause`, and `SolutionModifier`, we introduced a new `UsingClause`. The `UsingClause` expands to Rule 102 that adds the new keyword `USING` followed by a `SourceSelector` to define the name and parameters of the learning algorithm.

Semantics. According to [35], a SPARQL query consists of three parts: the *pattern matching part*, the *solution modifiers*, and the *output*. In that sense, the semantics of the `CREATE MINING MODEL` queries is the construction of new triples describing the metadata of the trained model (*i.e.*, SPARQL-ML introduces a new output type). An example of such metadata for the model induced in Listing 1.3 is shown in Listing 1.4, which follows the definitions of our *SPARQL Mining Ontology (SMO)* in Figure 6. The ontology enables to permanently save the parameters of a learned model, which is needed by the predict queries (see next section).

The ontology includes the model name, the used learning algorithm, all variables/features being used to train the classifier, as well as additional information, such as where to find the generated model file. In Listing 1.4, lines 1–11 show the constructed triples of a model with name *services*, while lines 13–28 show the metadata for two particular features of the model.

Step 2: Making Predictions Via Virtual Triple Patterns The second step to perform inductive reasoning with SPARQL-ML is to apply the previously induced model to draw conclusions about new samples from the population. After the induction of the model with the `CREATE MINING MODEL` statement, SPARQL-ML allows the user to make predictions via two new ARQ property

```
1  CREATE MINING MODEL <http://www.ifi.uzh.ch/services>
2   { ?service     RESOURCE TARGET
3     ?category    DISCRETE PREDICT
4                   { 'communication', 'economy',
5                     'education', 'food', 'medical',
6                     'travel', 'weapon' }
7     ?profile     RESOURCE
8     ?output      RESOURCE
9     ?outputType  RESOURCE
10    ?outputSuper RESOURCE
11    ?input       RESOURCE
12    ?inputType   RESOURCE
13    ?inputSuper  RESOURCE
14  }
15 WHERE
16  { ?service service:presents    ?profile ;
17             service:hasCategory ?category .
18
19    OPTIONAL
20     { ?profile profile:hasOutput     ?output .
21       ?output  process:parameterType ?outputType .
22
23       OPTIONAL
24        { ?outputType rdfs:subClassOf ?outputSuper . }
25     }
26
27    OPTIONAL
28     { ?profile profile:hasInput      ?input .
29       ?input   process:parameterType ?inputType .
30
31       OPTIONAL
32        { ?inputType rdfs:subClassOf ?inputSuper . }
33     }
34  }
35 USING <http://kdl.cs.umass.edu/proximity/rpt>
```

Listing 1.3. SPARQL-ML `CREATE MINING MODEL` query for semantic service classification. The goal of this query is to induce an RPT model that predicts the value for a service's `hasCategory` attribute that should be one out of `communication`, `economy`, `education`, `food`, `medical`, `travel`, and `weapon` as defined by the `DISCRETE PREDICT` keywords on line 4–5.

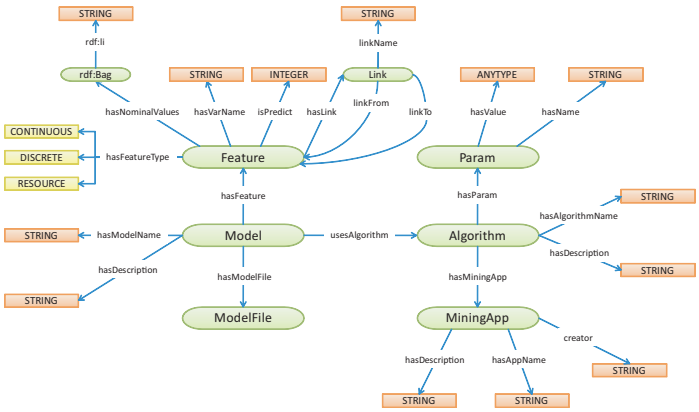

Fig. 6. SPARQL-ML Mining Ontology (SMO).

functions. In the following, these functions are called `sml:predict` and `sml:mappedPredict`.

Property functions are called whenever the predicate of a triple pattern is prefixed with a special name (*e.g.*, `sml`). In that case, a call to an external

```
<http://www.ifi.uzh.ch/ddis/services>
 a smo:Model ;
   smo:hasFeature
    <http://www.ifi.uzh.ch/ddis/services#output> ,
    <http://www.ifi.uzh.ch/ddis/services#category> ,
    <http://www.ifi.uzh.ch/ddis/services#input> ;
   smo:hasModelFile
    <http://www.ifi.uzh.ch/ddis/models/services.xml> ;
   smo:hasModelName  "services" ;
   smo:usesAlgorithm
    <http://kdl.cs.umass.edu/proximity/rpt> .

<http://www.ifi.uzh.ch/ddis/services#category>
 a smo:Feature ;
   smo:hasFeatureType  "DISCRETE" ;
   smo:hasNominalValues
    [ a rdf:Bag ;
        rdf:_1  "education" ;
        rdf:_2  "travel"
    ] ;
   smo:hasVarName  "category" ;
   smo:isPredict  "1" .

<http://www.ifi.uzh.ch/ddis/services#service>
 a smo:Feature ;
   smo:hasFeatureType  "RESOURCE" ;
   smo:hasVarName  "service" ;
   smo:isRootVar  "YES" .
```

Listing 1.4. Part of the metadata generated from inducing the RPT model as shown in Listing 1.3.

function is made and arguments are passed to the function (by the object of the triple pattern). For inductive reasoning, we are particularly interested in the following form of virtual triple pattern expressions:

$$\underbrace{(pred\ prob)}_{\text{subject}}\ \underbrace{predictionFunction}_{\text{predicate}}\ \underbrace{(arg1\ \ldots\ argN)}_{\text{object}}$$

In a nutshell, such pattern expressions define a list of arguments that are passed to a customized prediction function (CPF) by the object of the pattern expression. In our case, the first argument in this list (`arg1`) is a URI reference to the previously induced model that will be applied for making predictions. The rest of the arguments describe the new resource for which a prediction should be made.

Example 7 (SPARQL-ML Prediction Query) *Consider the SPARQL-ML query shown in Listing 1.5 that includes a single virtual triple pattern expression on lines 22–27. The goal of the query is to predict the value of a semantic service's* `hasCategory` *attribute by applying the previously induced model in Listing 1.3. The model is referenced by the model URI* `http://www.ifi.uzh.ch/ddis/services` *and passed as the first argument to the prediction function. The rest of the arguments define the attributes/features of the service that should be used for predicting its category. The result of the prediction (one out of* `communication`, `economy`, `education`, `food`, `medical`, `travel`, *or* `weapon`*), and its probability are finally bound on line 22 to the variables* `?prediction` *and* `?probability` *respectively.*

```
SELECT DISTINCT ?service ?prediction ?probability
WHERE
 { ?service service:presents ?profile .

   OPTIONAL
    { ?profile profile:hasOutput     ?output .
      ?output  process:parameterType ?outputType .

      OPTIONAL
       { ?outputType rdfs:subClassOf ?outputSuper . }
    }

   OPTIONAL
    { ?profile profile:hasInput      ?input .
      ?input   process:parameterType ?inputType .

      OPTIONAL
       { ?inputType rdfs:subClassOf ?inputSuper . }
    }

   PREDICTION
    { ( ?prediction ?probability )
          sml:predict
           ( <http://www.ifi.uzh.ch/ddis/services>
             ?service ?profile ?output ?outputType
             ?outputSuper ?input ?inputType
             ?inputSuper ) .
    }
 }
```

Listing 1.5. SPARQL-ML query to predict the the value of a service's `hasCategory` attribute.

[22]	*GraphPatternNotTriples*	::= OptionalGraphPattern | GroupOrUnionGraphPattern | GraphGraphPattern | PredictionBlockPattern
[22.1]	*PredictionBlockPattern*	::= 'PREDICTION' '{' ((Var1 FunctionCall)+ Filter?)+ '}'
[28]	FunctionCall	::= IRIref ArgList

Table 4. SPARQL-ML grammar rules for the `PREDICTION` statement.

Syntax and Grammar. The extended SPARQL-ML grammar for the prediction queries is shown in Table 4. To implement the virtual triple approach in SPARQL-ML, a new symbol called *PredictionBlockPattern* is added to the official SPARQL grammar rule of *GraphPatternNotTriples* [37]. The structure of *PredictionBlockPattern* resembles the one of *OptionalGraphPattern* but has completely different semantics: instead of matching patterns in the RDF graph, the triples in a *PredictionBlockPattern* act as virtual triple patterns that are interpreted by the query processor. A *PredictionBlockPattern* expands to Rule [22.1] that adds the new keyword `PREDICTION` to the grammar, which is followed by a number of virtual triples and optional `FILTER` statements.

Semantics. The semantics of a *PredictionBlockPattern* is basically that of a *prediction join*:[15] (1) the CPF maps the variables in the basic graph patterns of the query to the features in the specified model; (2) the CPF creates instances out of the mappings according to the induced model; (3) the model is used to classify an instance as defined in the `CREATE MINING MODEL` query; and (4), the values of the prediction and its probability are bound to variables in the predict query.

[15] `http://msdn2.microsoft.com/en-us/library/ms132031.aspx`

```
1 SELECT ?prediction ?probability
2 WHERE
3   { ?profile  profile:hasInput   ?input ;
4               profile:hasOutput  ?ouput .
5
6     ( ?prediction ?probability )
7         sml:predict ( <modelURI> ?profile
8                                  ?input ?output ) .
9   }
```

Listing 1.6. SPARQL-ML query exemplifying a prediction join operation

More formally, in the notation of Pérez [35], the semantics of a *PredictionBlockPattern* can be defined as follows. In [35], Pérez discussed four different SPARQL query types: join queries, union queries, optional queries, and filter queries. In accordance to [35], prediction joins are, thus, introduced as a new type of SPARQL queries for which the semantics is subsequently investigated in the remainder of this section. The new type is specified as follows (displayed in original SPARQL syntax on the left and algebraic syntax on the right):

Definition 4 (Prediction Join Query) *Prediction join queries involve basic graph patterns P and virtual graph patterns VP which trigger a call to a customized prediction function, i.e.,*

$$\{\ P\ \mathit{PREDICTION}\ \{\ VP\ \}\ \} \Longleftrightarrow (\ P\ \mathit{PREDJOIN}\ VP\ ).$$

Similarly to the definition of the join of ordinary sets of solution mappings, the prediction join of sets Ω_{BGP} and Ω_{VGP} can now be defined:

Definition 5 (Prediction Join Operation) *A prediction join $\bowtie_p$ of basic graph pattern expressions P and virtual graph pattern expressions VP extends the sets Ω_{BGP} from basic graph pattern matching with the sets of virtual solution mappings Ω_{VGP} from virtual graph pattern matching. The prediction join of Ω_{BGP} and Ω_{VGP} is defined as:*

$$\begin{aligned}\Omega_{BGP} \bowtie_p \Omega_{VGP} = \{\ \mu_1\ +\ \mu_2\ \mid\ & \\ & \mu_1 \in \Omega_{BGP},\ \mu_2 \in \Omega_{VGP},\ \mu_1,\ \mu_2\ \textit{are} \\ & \textit{compatible, and}\ 1 \leq card[\Omega_{VGP}](\mu_2) \leq 2\ \}\end{aligned}$$

Example 8 (Prediction Join Operation) *Consider the query shown in Listing 1.6 for the prediction of the value of the* `hasCategory` *attribute of a semantic service (assume an appropriate induction model was induced in an earlier step). Focusing only on service A, the evaluation of the basic triple patterns results in the set of solution mappings Ω_1, i.e.,*

$$\begin{aligned}\Omega_1 = \{\ \mu_{11}(?profile \mapsto SP1, ?input \mapsto \_CITY, \\ ?ouput \mapsto \_LUXURYHOTEL\}.\end{aligned}$$

The evaluation of the virtual triple pattern that specifies the property function for making predictions returns a set of virtual solution mappings Ω_2 that contains the values of the prediction and its probability. Assume the prediction model returns the following values, i.e.,

$$\Omega_2 = \{\ \mu_{21}(?prediction \mapsto travel, \\ ?probability \mapsto 0.99)\ \}.$$

Finally, the prediction join operation merges Ω_1 and Ω_2 into the set of solution mappings Ω_3:

$$\Omega_3 = \{\ \mu_{31}(?profile \mapsto SP1, ?input \mapsto \_CITY, \\ ?ouput \mapsto \_LUXURYHOTEL, \\ ?prediction \mapsto travel, ?probability \mapsto 0.99)\ \}.$$

In [27], the semantics of virtual graph patterns were defined as an evaluation function $[[vt]]$ that takes a virtual triple pattern vt and returns a virtual solution mapping μ_v. Adapting this equation to the inductive reasoning scenario in this paper, the evaluation of a SPARQL-ML predict query over a dataset D can be defined recursively as follows:

$$\begin{aligned} [[vt]] &= \{\ \mu_v(?v_1 \mapsto pre, ?v_2 \mapsto pro) \mid (pre, pro) \\ &= \text{pf:funct}\ (\ \mu(?x_1 \mapsto t_1), \ldots, \mu(?x_n \mapsto t_n)\ )\ \} \\ [[(P\ \text{PREDJOIN}\ VP)]]_D &= [[P]]_D \bowtie_p [[VP]] \end{aligned} \tag{2}$$

Again, the first part of Equation 2 takes a virtual triple pattern expression and returns a set of virtual solution mappings Ω_{VGP}. New solution mappings are generated that assign the value of a prediction and its probability to query variables (*i.e.*, $?v1$ and $?v2$) (note that Equation 2 only shows the case were both values are returned).

Pros and Cons. The following list summarizes the pros and cons of the virtual triple pattern approach to perform inductive reasoning with our SPARQL-ML framework.

+ A number of different prediction models can be used in the same query (which is useful to compare their performance).
+ The integration of inductive reasoning support into SPARQL provides an easy-to-use and flexible approach to quickly create and work with data mining models in SPARQL.
+ The values of the predictions and its probabilities are assigned to query variables, thus, can be reused in the query for filtering and ranking, or can be returned for arbitrary further processing.
+ Solution modifiers such as `ORDER BY` and `LIMIT` are applicable to the calculated prediction (probability) values.
+ A very simple adaption of `sml:predict` allows us to also apply the induced model on a dataset with a different ontology structure (*i.e.*, `sml:mappedPredict`).

Evaluation/Validation Task	Dataset(s)
Business Project Success Experiment	synthetic business project dataset
Semantic Web Service Classification Experiment	OWL-S TC v2.1
SVM-Benchmark Experiment	SWRC/AIFB dataset
Bug Prediction Experiment	Eclipse `updateui`, `updatecore`, `search`, `pdeui`, `pdebuild`, and `compare` plug-ins

Table 5. The four tasks and datasets we considered to evaluate/validate our novel inductive reasoning extension.

- The virtual triple pattern expressions we use for prediction are somehow 'overloaded' (*i.e.*, the property functions potentially have a long parameter list). Furthermore, the functions may return a list of prediction-probability values.
- The SPARQL grammar needs to be extended to account for the `PREDICTION` statements (which requires an adaptation of the query engines).
- Queries using property functions depend on a query engine extension currently only implemented in Jena ARQ and, hence, have limited interoperability.

4 Evaluation/Validation of SPARQL-ML

Our inductive reasoning method presented in Section 3 relies on statistical induction to reason over Semantic Web data. We have implemented inductive reasoning as an extension to the RDF query language SPARQL. More specifically, we use virtual triple patterns as key technology to integrate inductive reasoning with the traditional Semantic Web infrastructure.

This section is devoted to the application and evaluation of this novel reasoning method for three Semantic Web and one Software Analysis task. These tasks along with the datasets we used to evaluate them are listed in Table 5. In the following, we will briefly give an overview of each of these tasks.

Business Project Success Experiment. In order to show the ease-of-use and predictive capability of our inductive reasoning framework SPARQL-ML, we put together a proof of concept setting with a small, artificially created dataset. To that end, in our first experiment in Section 4.1, we show that using a synthetic dataset, the combination of statistical inference with logical deduction produces superior performance over statistical inference only.

Semantic Web Service Classification Experiment. The goal of our semantic service classification experiment in Section 4.2 is to evaluate our novel inductive reasoning extension to the task of performing automatic service classification. To that end, we perform a Semantic Web service category prediction experiment (*i.e.*, automatically generate semantic annotation/metadata for se-

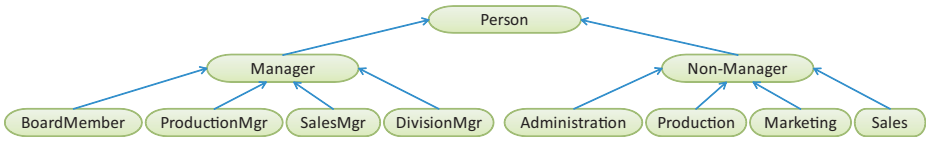

Fig. 7. Example business ontology.

mantic services). As benchmarking dataset, we use a large OWL-S semantic service retrieval test collection.

SVM-Benchmark Experiment. In our third experiment—the SVM-benchmark experiment—we compare the prediction performance of our SPARQL-ML approach to another state-of-the-art kernel-based Support Vector Machine (SVM) [6] using a real-world data set.

Software Bug Prediction Experiment. Finally, in our bug prediction experiment in Section 4.4, we aim to show some of the advantages of inductive reasoning for *Software Analysis*. Specifically, we will use SPARQL-ML in combination with the EvoOnt software model to perform bug prediction. To that end, the defect location experiment presented in [3] is repeated.

4.1 Business Project Success Experiment

Evaluation Methodology and Dataset. The synthetic *business project dataset* consists of different business projects and the employees of an imaginary company. The company has 40 employees each of which having one out of 8 different occupations. Figure 7 shows part of the created ontology in more detail. In our dataset, 13 employees belong to the superclass `Manager`, whereas 27 employees belong to the superclass `Non-Manager`.

We then created business projects and randomly assigned up to 6 employees to each project. The resulting teams consist of 4 to 6 members. Finally, we randomly defined each project to be successful or not, with a bias for projects being more successful, if more than three team members are of type `Manager`. The resulting dataset contains 400 projects with different teams. The prior probability of a project being successful is 35%. We did a 50:50 split of the data and followed a single holdout procedure, swapping the roles of the testing and training set and averaged the results.

Experimental Results. Listing 1.7 shows the `CREATE MINING MODEL` query that we used in the model learning process. We tested different learning algorithms with and without the support of inferencing. With the reasoner disabled, the last triple pattern in the `WHERE` clause (line 10) matches only the direct type of the received employee instance (*i.e.*, if an employee is a 'direct' instance of class `Manager`). This is the typical situation in relational databases without the support of inheritance. With inferencing enabled, the last triple pattern also matches all inferred types, indicating if an employee is a `Manager` or not.

Given the bias in the artificial dataset, it is to be expected that the ability to infer if a team member is a `Manager` or not is central to the success of the

```
1  CREATE MINING MODEL <http://www.example.org/projects>
2   { ?project RESOURCE TARGET
3     ?success DISCRETE PREDICT {'Yes','No'}
4     ?member  RESOURCE
5     ?class   RESOURCE
6   }
7  WHERE
8   { ?project ex:isSuccess ?success .
9     ?project ex:hasTeam   ?member .
10    ?member  rdf:type     ?class .
11  }
12 USING <http://kdl.cs.umass.edu/proximity/rpt>
```

Listing 1.7. SPARQL-ML `CREATE MINING MODEL` query. The goal of this query is to induce an RPT model that predicts the value for a project's `isSuccess` attribute that should be either `Yes` or `No` as defined by the `DISCRETE PREDICT` keywords on line 3.

induction procedure. Consequently, we would expect that models induced on the inferred model should exhibit a superior performance. The results shown in Figure 8 confirm our expectations. The Figure shows the results in terms of prediction accuracy (ACC; in legend), Receiver Operating Characteristics (ROC; graphed), and the area under the ROC-curve (AUC; also in legend). The ROC-curve graphs the true positive rate (y-axis) against the false positive rate (x-axis), where an ideal curve would go from the origin to the top left (0,1) corner, before proceeding to the top right (1,1) one [36]. It has the advantage to show the prediction quality of a classifier independent of the distribution (and, hence, prior) of the underlying dataset. The area under the ROC-curve is, typically, used as a summary number for the curve. Note that a random assignment whether a project is successful or not is also shown as a line form the origin (0,0) to (1,1). The learning algorithms shown are a Relational Probability Tree (RPT), a Relational Bayes Classifier (RBC), both with and without inferencing, and, as a baseline, a k-nearest neighbor learning algorithm (k-NN) with inferencing and $k = 9$ using a maximum common subgraph isomorphism metric [44] to compute the closeness to neighbors.

As the Figure shows, the relational methods clearly dominate the baseline k-NN approach. As expected, both RPT and RBC with inferencing outperform the respective models without inferencing. It is interesting to note, however, that RPTs seem to degrade more with the loss of inferencing than RBCs. Actually, the lift of an RBC with inferencing over an RBC without inferencing is only small. These results support our assumption that the combination of induction and deduction should outperform pure induction. The major limitation of this finding is the artificial nature of the dataset. We, therefore, decided to conduct further experiments with the same goals using real-world datasets, which we present in the following sections.

4.2 Semantic Web Service Classification Experiment

In this section, we proceed with the evaluation of SPARQL-ML on a real-world dataset. Specifically, we show how SPARQL-ML can be used to automatically classify Semantic Web services into their most appropriate service *category*. Ac-

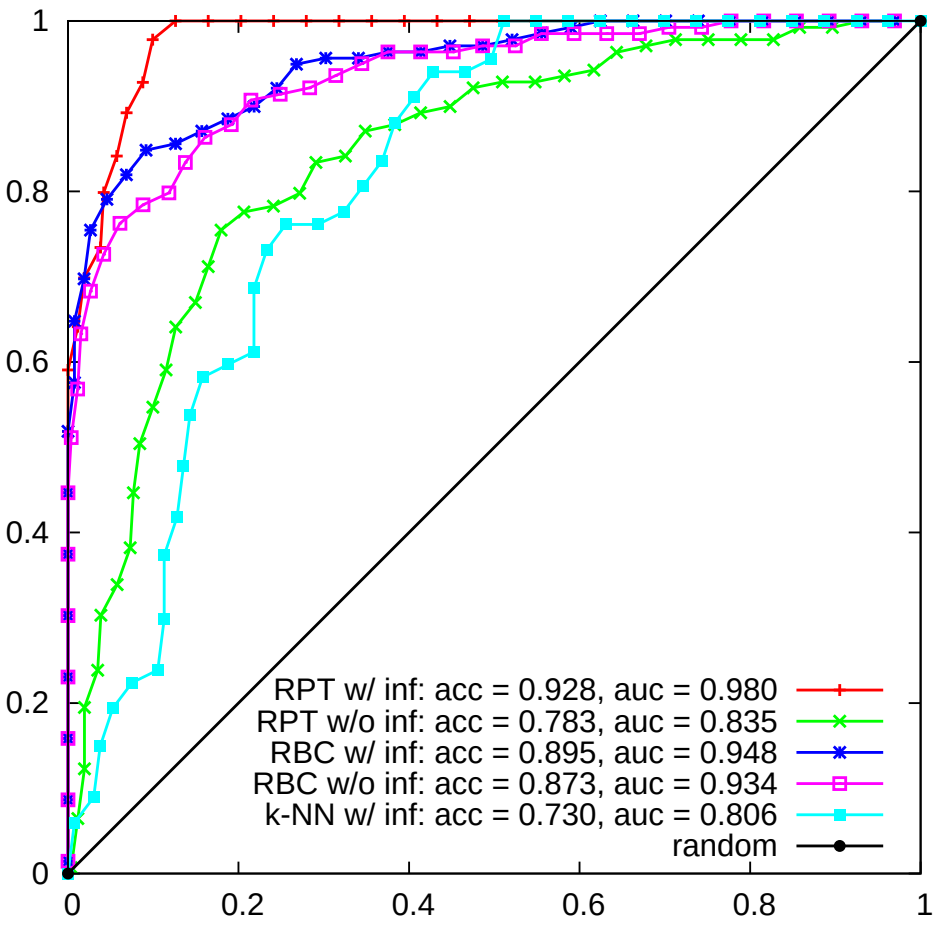

Fig. 8. ROC-Curves of business project success prediction.

cording to [22], a web service category describes the general kind of service that is offered, such as "travel services" or "educational services".

In a nutshell, our SPARQL-ML framework is used to classify/predict the category of a semantic service, which is usually a string value, say, `travel` or `education`. This value can then be used to *tag* (annotate) the semantic service.[16]

Evaluation Methodology and Dataset. For all our service classification experiments we use the OWLS-TC v2.1 Semantic Web service retrieval test collection.[17] OWLS-TC contains 578 semantic service descriptions of seven different categories. These categories are `economy`, `education`, `travel`, `medical`,

[16] *E.g.*, in Semantic Web terminology add a new triple to the service description holding the value of the classification step. Note, however, that our focus clearly lies on service classification rather than service annotation

[17] `http://projects.semwebcentral.org/projects/owls-tc/`

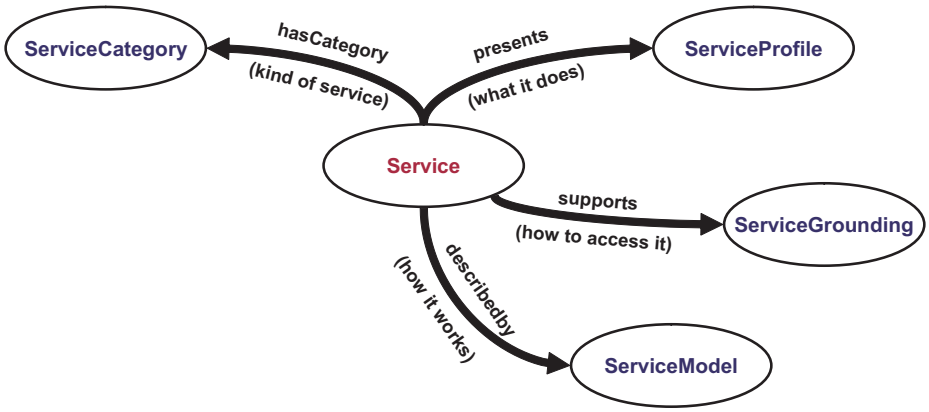

Fig. 9. Extended OWL-S upper ontology model. In addition to the service profile, grounding, and model, an extra relation to the service category is added to the description of a service.

`communication`, `food`, and `weapon`. The prior distribution of the services is `economy` = 35.63%, `education` = 23.36%, `travel` = 18.34%, `medical` = 8.99%, `communication` = 5.02%, `food` = 4.33%, and `weapon` = 4.33% (*i.e.*, `economy` is the category with the most services).

In order to predict a semantic service's `hasCategory` attribute, we first had to assert this information in the dataset (as it is originally not). In other words, we had to extend the OWL-S service ontology model with an additional relation to the service category. The extended OWL-S ontology is shown in Figure 9.

Using these extended service descriptions, we are able to write `CREATE MINING MODEL` queries that (i) define the instances to be use for model induction and (ii) specify the learning algorithm and its parameters. Note that in all our experiments we limited our investigations to the I/O concepts of services as we believe that they are most informative for this task (cf. [20]).

Experimental Results. Listing 1.3 shows the `CREATE MINING MODEL` query that we used in the model learning step. By using `OPTIONAL` patterns, we enable the inclusion of services with no outputs or inputs. The additional `OPTIONAL` pattern for the `rdfs:subClassOf` triple enables us to run the same query on the asserted and the inferred data.

We ran the experiment once on the asserted and once on the (logically) inferred model using the predict query shown in Listing 1.5. Furthermore, we performed a 10-fold cross validation where 90% of the data was used to learn a classification model and the remaining 10% to test the effectiveness of the learned model, which is standard practice in machine learning (see [45]). For our experiments, we induced a RPT to predict the service category of a service based on its input and output concepts. We chose an RPT because in all our experiments it turned out to perform superior than RBCs.

Category	FP Rate		Precision		Recall		F-measure	
	w/o inf	*w/ inf*	*w/o inf*	*w/ inf*	*w/o inf*	*w/ inf*	*w/o inf*	*w/ inf*
communication	0.007	0.004	0.819	0.900	0.600	0.600	0.693	0.720
economy	0.081	0.018	0.810	0.964	0.644	0.889	0.718	0.925
education	0.538	0.090	0.311	0.716	0.904	0.869	0.463	0.786
food	0	0.002	0	0.960	0	0.800	0	0.873
medical	0.006	0.030	0	0.688	0	0.550	0	0.611
travel	0	0.069	1	0.744	0.245	0.873	0.394	0.803
weapon	0.002	0.002	0.917	0.964	0.367	0.900	0.524	0.931
average	**0.091**	**0.031**	**0.551**	**0.848**	**0.394**	**0.783**	**0.399**	**0.807**
t-test (paired, one-tailed)	**p=0.201**		**p=0.0534**		**p=0.00945**		**p=0.0038**	

Table 6. Detailed results for the Semantic Web service classification experiments. As can be observed, the models induced on the (logically) inferred I/O concepts (*w/ inf*) perform considerably better than the ones induced on only the asserted information (*w/o inf*) across almost all measures and categories.

The averaged classification accuracy of the results of the 10 runs is 0.5102 on the asserted and 0.8288 on the inferred model. Hence, the combination of logical deduction with induction improves the accuracy by 0.3186 over pure induction. The detailed results of our experiments are shown in Table 6 that further confirm this result for all seven categories by listing the typical data mining measures false positive rate (FP rate), precision, recall, and F-measure for all categories. As the results of the t-test show, the differences for recall and F-measure are (highly) significant. The results for precision just barely misses significance at the 95% level.

When investigating the structure of the RPTs, the trees induced on the inferred model clearly exploit inheritance relations using the transitive *rdfs:subClassOf* property, indicating that the access to the newly derived triples improves the determination of a service's category. The SRL algorithms are able to exploit the richer relational neighborhood to improve their performance. These observations further support our finding that a combination of deduction and induction is useful for Semantic Web tasks and can be easily achieved with SPARQL-ML.

4.3 SVM-Benchmark Experiment

Evaluation Methodology and Dataset. With our third set of experiments, we aimed to show possible advantages of SPARQL-ML over another state-of-the-art method. Specifically, we compared the off-the-shelf performance of a simple **xx**-lines SPARQL-ML statement (see Listing 1.8) with a Support Vector Machine (SVM) based approach proposed by Bloehdorn and Sure [7] following exactly their evaluation procedure.[18] In their work, they introduced a framework for the design and evaluation of kernel methods that are used in Support Vector Machines, such as SVM^{light} [24]. The framework provides various kernels for the comparison of classes as well as datatype and object properties of instances. Moreover, it is possible to build customized, weighted combinations of such kernels. Their evaluations include two tasks: (1) prediction of the affiliation

[18] We would like to thank them for sharing the exact dataset used in their paper.

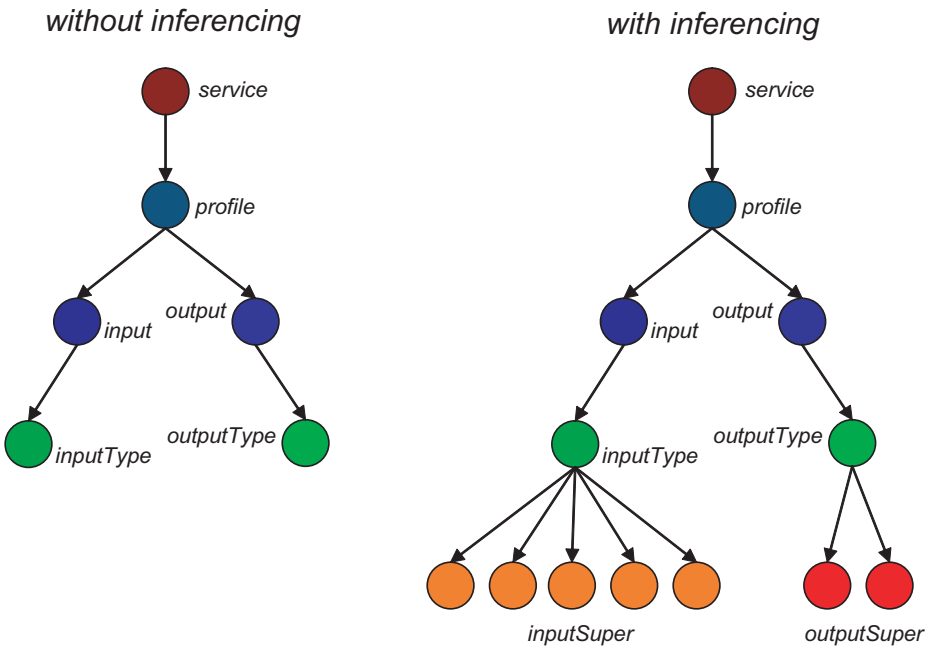

Fig. 10. Service subgraphs built for the Semantic Web service classification task, on the left without inferencing and on the right with inferencing.

a person belongs to (`person2affiliation`), and (2) prediction of the affiliation a publication is related to (`publication2affiliation`). As a dataset they used the SWRC ontology—a collection of OWL annotations for persons, publications, and projects, and their relations from the University of Karlsruhe.[19]

In order to understand the course of our experiment, we think a few words about the experimental procedure described in [7] are necessary. For each of the two tasks, Bloehdorn and Sure performed exactly four binary classification experiments and averaged the results for each task. More precisely, consider the `person2affiliation` task: for each of the four distinct research groups and 177 persons in the SWRC dataset, the authors conducted a two-class classification experiment to predict whether a person belongs to a research group or not. The same approach was chosen for the `publication2affiliation` task: for each of the four research groups and 1232 publication instances in the dataset, a binary classification experiment was performed in order to predict whether one of the authors of the publication is affiliated with the group.

In order to perform the identical experiment as described in [7], we first had to add the information about a person's affiliation to the dataset via a couple of `belongsToGroupX` ($X = 1 \ldots 4$) datatype properties. This was necessary because we wanted to predict the value of this property (either 'Yes' or 'No') using our

[19] `http://ontoware.org/projects/swrc/`

person2affiliation					publication2affiliation				
algorithm	*err*	*prec*	*rec*	*F-measure*	*algorithm*	*err*	*prec*	*rec*	*F-measure*
sim-ctpp-pc, c=1	4.49	95.83	58.13	72.37	sim-cta-p, c=10	0.63	99.74	95.22	97.43
RBC w/o inf	9.43	79.41	77.94	78.51	RBC w/o inf	1.035	97.36	94.21	95.68
RBC w/ inf	9.39	80.90	75.46	77.73	RBC w/ inf	0.73	95.53	97.52	96.46

Table 7. LOOCV results for the *person2affiliation* and *publication2affiliation* tasks.

```
CREATE MINING MODEL <http://example.org/svm>
 { ?person           RESOURCE  TARGET
   ?value            DISCRETE  PREDICT  {'Yes','No'}
   ?personType       RESOURCE
   ?project          RESOURCE
   ?topic            RESOURCE
   ?publication      RESOURCE
   ?publicationType  RESOURCE
  }
WHERE
 { ?person  swrc:affiliation     ?affiliation ;
            rdf:type             ?personType ;
            uzh:belongsToGroup1  ?value .

   OPTIONAL
    { ?person  swrc:worksAtProject  ?project . }
   OPTIONAL
    { ?topic  swrc:isWorkedOnBy  ?person . }
   OPTIONAL
    { ?person       swrc:publication  ?publication .
      ?publication  rdf:type          ?publicationType .
    }
 }
USING <http://kdl.cs.umass.edu/proximity/rbc>
```

Listing 1.8. `CREATE MINING MODEL` query for the person2affiliation task.

proposed SPARQL-ML SRL methods. An example `CREATE MINING MODEL` query is shown in Listing 1.8, where the goal is to predict whether a person belongs to the research group `Group1`. We ran this query exactly four times with different `belongsToGroupX` properties, recorded the results, and averaged them.

Experimental Results. Table 7 summarizes the macro-averaged results that were estimated via Leave-One-Out Cross-Validation (LOOCV). We applied both, an RBC and an RPT learning algorithm to both tasks. The table also reports the best-performing SVM results from Bloehdorn and Sure's experiments. The RBC clearly outperformed the RPT in both predictions, hence, we report only on the results given by the RBC. For both tasks the performance of the inferred model is not very different from the one induced on the asserted model. When consulting Listing 1.8 (for `person2affiliation`) it is plausible to conclude that the only inferred properties (types of persons and publications) do not help to classify a person's or a publication's affiliation with an organizational unit.

For the `person2affiliation` task, Table 7 shows that our method clearly outperforms the kernel-based approach in terms of recall, but has only marginally better F-Measure improvement. This is because our method is clearly inferior in terms of prediction error and precision. For the `publication2affiliation` task, the results are even worse: turning the reasoner on improves, at least, the results compared to no reasoner used, however, the results are still inferior compared to the kernel-based approach by Bloehdorn and Sure across all performance measures.

person2affiliation					publication2affiliation				
algorithm	*err*	*prec*	*rec*	*F-measure*	*algorithm*	*err*	*prec*	*rec*	*F-measure*
sim-ctpp-pc, c=1	4.49	95.83	58.13	72.37	sim-cta-p, c=10	0.63	99.74	95.22	97.43
RBC w/o inf	3.53	87.09	80.52	83.68	RBC w/o inf	0.09	98.83	99.61	99.22
RBC w/ inf	3.67	85.72	80.18	82.86	RBC w/ inf	0.15	97.90	99.25	98.57

Table 8. LOOCV results for the *person2affiliation* and *publication2affiliation* tasks.

```
1  CREATE MINING MODEL <http://example.org/svm>
2   { ?person           RESOURCE  TARGET
3     ?affiliation      DISCRETE  PREDICT  {'ID1','ID2','ID3','ID4'}
4     ?personType       RESOURCE
5     ?project          RESOURCE
6     ?topic            RESOURCE
7     ?publication      RESOURCE
8     ?publicationType  RESOURCE
9    }
10 WHERE
11  { ?person  swrc:affiliation     ?affiliation ;
12             rdf:type             ?personType .
13
14    OPTIONAL
15     { ?person  swrc:worksAtProject  ?project . }
16    OPTIONAL
17     { ?topic  swrc:isWorkedOnBy  ?person . }
18    OPTIONAL
19     { ?person       swrc:publication  ?publication .
20       ?publication  rdf:type          ?publicationType .
21     }
22  }
23 USING <http://kdl.cs.umass.edu/proximity/rbc>
```

Listing 1.9. `CREATE MINING MODEL` query for the person2affiliation task.

Because these results were not very promising, we asked ourselves how we could achieve better prediction performance. We thought, why not perform a real multi-class prediction experiment instead of four rather tedious individual experiments and averaging the results. Luckily, with our SPARQL-ML approach we are able to perform exactly this kind of prediction experiment. The corresponding example query is shown in Listing 1.9 and the results in Table 8. Note that this query can use the ontology *as is*, *i.e.*, the dataset does not have to be extended with additional relations (as was the case in Listing 1.8).

As Table 8 clearly shows, our multi-class prediction method outperforms the kernel-based approach in terms of prediction error, recall, and F-Measure, while having an only slightly lower precision. The slightly lower precision could be a result of the limitation to just a few properties used by an off-the-shelf approach without a single parameter setting, whereas the SVM approach is the result of extensive testing and tuning of the kernel method's properties and parameters.

We conclude from this experiment, that writing a SPARQL-ML query is a simple task for everyone familiar with the data and the SPARQL-ML syntax. Kernels, on the other hand, have the major disadvantage that the user has to choose from various kernels, kernel modifiers, and parameters. This constitutes a major problem for users not familiar with kernels and SVM algorithms.

4.4 Bug Prediction Experiment

In our last experiment, we evaluate the applicability and usefulness of our novel inductive reasoning framework SPARQL-ML for *bug prediction*. We, therefore, evaluated the predictive power of our SPARQL-ML approach on several real-world software projects modeled in the EvoOnt format (see [28]). To that end, we will compare the off-the-shelf performance of SPARQL-ML with a traditional, propositional data mining approach proposed in [3] following exactly their evaluation procedure. To achieve this goal, we use *historical/evolutionary information* about the software projects in all our experiments. This information is provided by a concurrent versions system (CVS) and a bug-tracking system (*i.e.*, Bugzilla).[20]

Evaluation Methodology and Datasets. The data used in this case study was collected from six plug-ins of the Eclipse open source project in the overall time span from January 3, 2001 to January 31, 2007.[21] The plug-ins are `compare`, `pdebuild`, `pdeui`, `search`, `updatecore`, and `updateui`, which are all available at the CVS repository at `dev.eclipse.org`.

In a nutshell, the experimental procedure can be summarized as follows: first, along with the data from CVS and Bugzilla, we exported each of the plug-ins into our EvoOnt format; second, we created a small extension to EvoOnt to take into account the 22 extra features from [3] that are used for model induction and making predictions; and third, we wrote SPARQL-ML queries for the induction of a mining model on the training set as well as for the prediction of bugs on the test set. The queries in Listings 1.10 and 1.11 show an example of the `CREATE MINING MODEL` and `PREDICT` statements we used for the model induction and prediction tasks respectively.

Addressing the first step, exporting the information from CVS and Bugzilla into our EvoOnt format, the information from the first releases up to the last one released in January 2007 was considered. For the second step, the extension of the EvoOnt model with the additional features for learning and predicting, we exploited the fact that EvoOnt (and more generally, the OWL data format) is easily extendable with additional classes and properties. We had to extend EvoOnt with a total number of 22 additional features, which were all computed in a preprocessing step and added to the OWL class `File` in EvoOnt's Version Ontology Model (VOM) via a set of new OWL datatype properties (*e.g.*, `vom:loc`, `vom:lineAddedIRLAdd`, etc.). Furthermore, for each ontologized file of the plug-ins, an additional `vom:hasError` property is added. The value of the property is either 'Yes' or 'No' depending on wether the file was mentioned in a bug report from Bugzilla.

In the experiments in [3], six different models were trained using Weka's J48 decision tree learner. The first model does not take into account any temporal features whilst the second to fifth model all use a variation of different temporal and non-temporal features for model induction. Finally, the sixth model

[20] `http://www.bugzilla.org/`
[21] `http://www.eclipse.org/`

```
1  CREATE MINING MODEL <http://www.example.org/bugs>
2   { ?file                    RESOURCE   TARGET
3     ?error                   DISCRETE   PREDICT
4                                           {'YES','NO'}
5     ?lineAddedIRLAdd         CONTINUOUS
6     ?lineDeletedIRLDel       CONTINUOUS
7     ?revision1Month          CONTINUOUS
8     ?defectAppearance1Month  CONTINUOUS
9     ?revision2Months         CONTINUOUS
10    ?reportedIssues3Months   CONTINUOUS
11    ?reportedIssues5Months   CONTINUOUS
12  }
13 WHERE
14  { ?file     vom:hasRevision  ?revision .
15    ?revision vom:creationTime ?creation .
16    FILTER ( xsd:dateTime(?creation)
17             <= "2007-01-31T00:00:00"^^xsd:dateTime )
18
19    ?file      vom:hasError ?error .
20
21    OPTIONAL { ?file vom:lineAddedIRLAdd
22                       ?lineAddedIRLAdd . }
23    OPTIONAL { ?file vom:lineDeletedIRLDel
24                       ?lineDeletedIRLDel . }
25    OPTIONAL { ?file vom:revision1Month
26                       ?revision1Month . }
27    OPTIONAL { ?file vom:defectAppearance1Month
28                       ?defectAppearance1Month . }
29    OPTIONAL { ?file vom:revision2Months
30                       ?revision2Months . }
31    OPTIONAL { ?file vom:reportedIssues3Months
32                       ?reportedIssues3Months . }
33    OPTIONAL { ?file vom:reportedIssues5Months
34                       ?reportedIssues5Months . }
35  }
36 USING <http://kdl.cs.umass.edu/proximity/rpt>
```

Listing 1.10. SPARQL-ML `CREATE MINING MODEL` query to induce a model using the most significant code features from [3].

is a *summary model* that uses only those features that turned out to be most significant/discriminant in the other models.

For each set of discriminating features, we created a `CREATE MINING MODEL` query to induce a model using either a Relational Probability Tree or a Relational Bayesian Classifier as prediction algorithm. Listing 1.10 shows the corresponding SPARQL-ML query for inducing a model using only the most significant features from [3]. For model induction, all the files of the plug-ins that were released before January 31, 2007 are considered (lines 16–17). Variable `?file` is the target variable that is linked to variable `?error` for which a prediction should be made (either 'Yes' or 'No') expressing if the file is likely to be error-prone or not (lines 2 and 3). Finally, the induced model is available for predictions via its model URI `<http://www.example.org/bugssignificant>`.

To test the model, we applied the predict query shown in Listing 1.11. The query first selects the source code files for which a revision was made before January 31, 2007 (line 6), and second, applies the previously induced model to classify a file as either buggy or non-buggy (lines 24–32).[22] The result of the prediction and its probability are finally bound on line 25 to the variables `?prediction` and `?probability`. [23]

[22] Note that every file we considered has at least one revision (*i.e.*, for when it was created/checked into CVS).

[23] Furthermore note that the prediction query in Listing 1.11 is only shown for illustration purposes. This kind of query is useful to predict if a *new, unseen* file is likely to

```
SELECT DISTINCT ?file ?prediction ?probability
WHERE
 { ?file     vom:hasRevision  ?revision .
   ?revision vom:creationTime ?creation .

   FILTER ( xsd:dateTime ( ?creation )
            <= "2007-01-31T00:00:00"^^xsd:dateTime )

   OPTIONAL { ?file vom:lineAddedIRLAdd
                    ?lineAddedIRLAdd . }
   OPTIONAL { ?file vom:lineDeletedIRLDel
                    ?lineDeletedIRLDel . }
   OPTIONAL { ?file vom:revision1Month
                    ?revision1Month . }
   OPTIONAL { ?file vom:defectAppearance1Month
                    ?defectAppearance1Month . }
   OPTIONAL { ?file vom:revision2Months
                    ?revision2Months . }
   OPTIONAL { ?file vom:reportedIssues3Months
                    ?reportedIssues3Months . }
   OPTIONAL { ?file vom:reportedIssues5Months
                    ?reportedIssues5Months . }

   PREDICT
    { ( ?prediction ?probability )
          sml:predict
            ( <http://www.example.org/bugs>
              ?file ?lineAddedIRLAdd ?lineDeletedIRLDel
              ?revision1Month ?defectAppearance1Month
              ?revision2Months ?reportedIssues3Months
              ?reportedIssues5Months ) .
    }
 }
```

Listing 1.11. SPARQL-ML predict query to classify a source code file as either buggy or non-buggy.

Experimental Results. The results of the bug prediction experiments are summarize in Figures 11 and 12 that illustrate the performance of the temporal and non-temporal feature models using RPTs and RBCs. The results are again presented in terms of prediction accuracy (acc; in legend), Receiver Operating Characteristics (ROC; graphed), and the area under the ROC curve (auc; also in legend).

Figures 11 and 12 show the performance of the best model from [3] as a baseline (the black line with bullet points; acc = 0.992, auc = 0.925). This is the model that was trained with only the most significant/discriminating features. As can be seen, the best SRL model is the RBC model induced on the 3-months features (auc = 0.977), closely followed by the RPT model on only the most significant features from [3] (auc = 0.972). It can be observed, that with the exception of the RPT model for the most significant features, all the RBC models slightly outperform the RPT models in terms of area under the curve. Examining accuracy, the RPT models, on the other hand, outperform the RBC models.

Furthermore, is is interesting to observe that all but the models trained on the 1-month features outperform the traditional, propositional learning approach of [3] in terms of area under the curve. For both the RPT and RBC algorithm, the 1-month model shows the worst performance compared with the baseline as well as with the rest of the temporal/non-temporal feature models. This is

be buggy or not. However, as we use the same set of files for training and testing, we currently run a variation of the scripts proposed in [23] (pages 102–108) to perform cross-validation.

contrary to the findings of [3] where the 1-month model was second best in terms of accuracy and at third position for auc.

The traditional model is, however, better in terms of prediction/classification accuracy (acc = 0.992). Note that the use of accuracy as a measure for the quality of the prediction is, however, misleading as it does not relate the prediction to the prior probability of the classes (*i.e.*, 'Yes'/'No' for the value of `vom:hasError`). As pointed out in [3], this is especially problematic in datasets which are heavily skewed (*i.e.*, that have a distribution of values far from being normal). As shown by the authors, the bug prediction dataset is indeed heavily skewed with a total number of 3691 non-buggy and 14 buggy classes. Hence, as mentioned earlier, the ROC curves and the area under the curve are more meaningful measures as they provide a prior-independent approach for comparing the quality of predictors.

Last but not least, note that the best performing RPT/RBC models (significant features for RPT, 3-months features for RBC) also have the highest prediction/classification accuracy among the SRL models (acc = 0.985 and acc = 0.977).

5 Discussion and Limitations

We briefly discuss some of the limitations of our novel inductive reasoning approach. SPARQL-ML's major drawback is the use of virtual triple patterns that some might deem as conceptually problematic. However, in this work we regard virtual triple patterns simply as part of the inferred knowledgebase; in other words, the specification of a prediction function is akin to the specification of an additional inferencing rule. Another limitation of the virtual triple pattern approach lies, of course, in the need for extending existing SPARQL query engines with the necessary language statements.

Regarding semantic service classification, the performance of the prediction/-classification task might heavily depend on the expressiveness of the used ontologies. The Semantic Web services used in our experiments define their I/O concepts using extensive (*i.e.*, deep) ontologies (*e.g.*, the *portal.owl* and *travel.owl* ontologies), which enables to derive extensive, additional knowledge about the I/Os. Using ontologies with flatter inheritance structures will, therefore, likely result in inferior results. We note, however, that this performance loss is a limitation of the used ontologies and not of the SRL algorithms themselves. Therefore, we speculate that the loss could be eliminated by using more comprehensive ontologies.

Regarding our bug prediction experiments, we note as a technical limitation that we are currently not able to perform cross-validation through the query engine. Thus, if we want to use the same dataset for training and testing, we currently have to use specialized scripts for making predictions and calculating the performance measures.

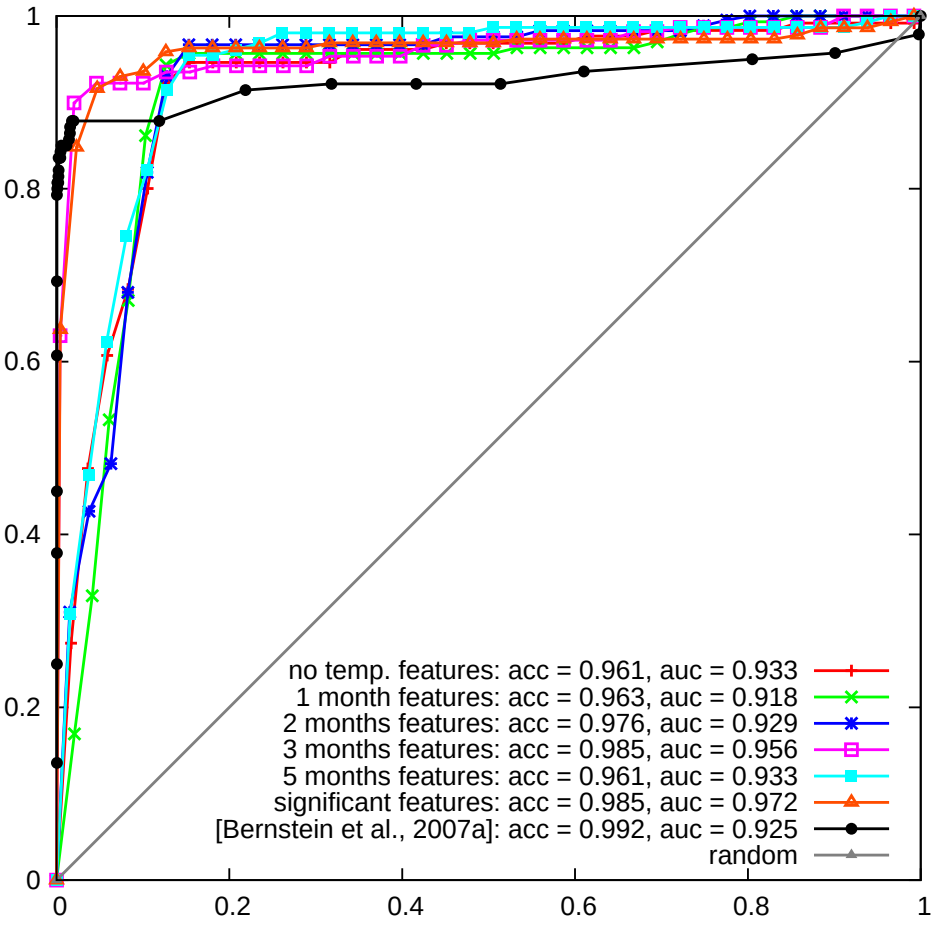

Fig. 11. ROC curves for all of the temporal and non-temporal models of the bug prediction experiments using RPTs. The model induced on the most significant features reported in [3] outperforms the baseline (black line) as well as all the other RPT models in terms of area under the curve.

6 Conclusions and Perspectives

In this paper, we have introduced our novel inductive reasoning extension to the Semantic Web. This extension aims at complementing the classical deductive description logic reasoning facilities of the traditional Semantic Web infrastructure (*i.e.*, it allows us to draw conclusions from the asserted facts in a knowledgebase which are otherwise *not* deducible by the classical approaches). Our extension is tightly integrated with the RDF query language SPARQL, providing access to the newly derived knowledge through the query engine. To that end, our extension exploits SPARQL virtual triple patterns that perform pattern matching by

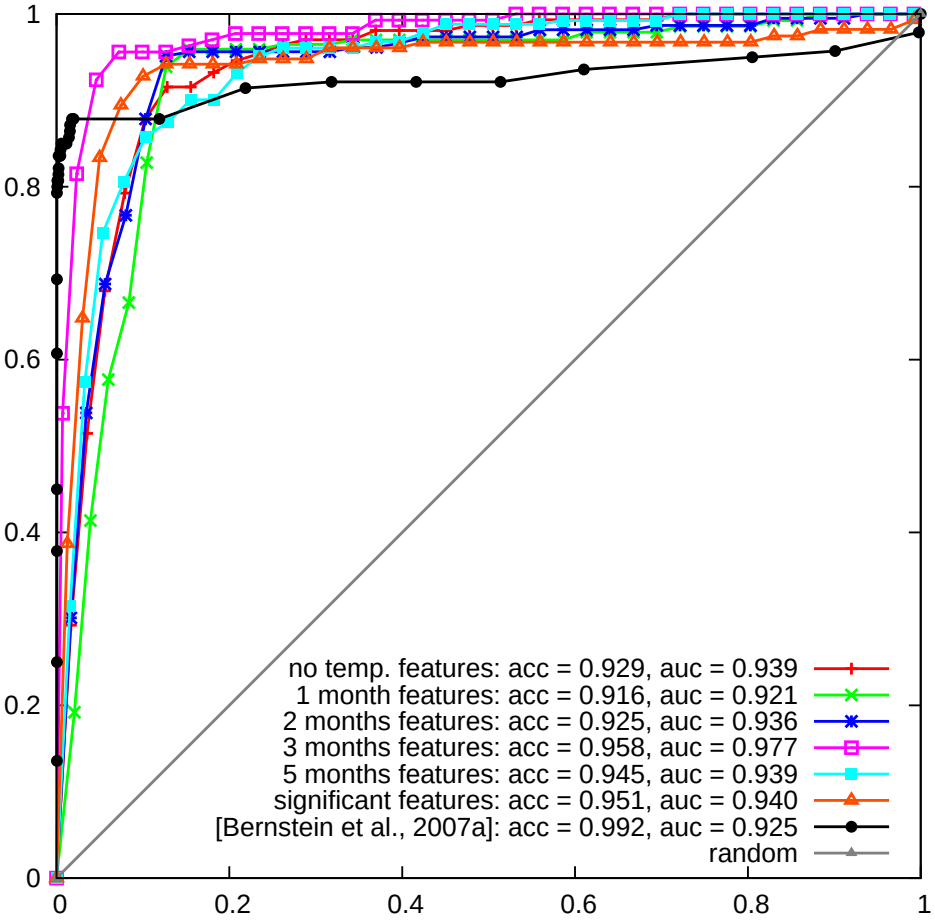

Fig. 12. ROC curves for all of the temporal and non-temporal models of the bug prediction experiments using RBCs. The 3-months feature model outperforms all the other models (including the baseline) in terms of area under curve.

calling a customized, external piece of code, rather than matching triple patterns against an RDF graph.

To evaluate/validate our novel extension, we performed four sets of experiments using synthetic and real-world datasets. In our first case study, we fully analyzed SPARQL-ML on a synthetic dataset to show its excellent prediction/-classification quality in a proof-of-concept setting. Secondly, we have shown the benefits of Statistical Relational Learning (SRL) algorithms (particularly Relational Probability Trees) to perform Semantic Web service classification using a well-known Semantic Web benchmarking dataset.

By enabling/disabling ontological inference support in our experiments, we came to the conclusion that the combination of statistical inference with logical

deduction produces superior performance over statistical inference only. These findings support our assumption that the interlinked Semantic Web data is a perfect match for SRL methods due to their focus on relations between objects (extrinsic attributes) in addition to features/attributes of objects of traditional, propositional learning techniques (intrinsic attributes).

In our third set of experiments, we have shown SPARQL-ML's superiority to another related, kernel-based approach used in Support Vector Machines. Finally, in the bug prediction case study, we have demonstrated, that inductive reasoning enabled by our SPARQL-ML framework allows us to easily perform bug prediction on semantically annotated software source code. Our empirical findings suggest that SPARQL-ML is indeed able to predict bugs with a very good accuracy, which, ultimately makes SPARQL-ML a suitable tool to help improve the quality of software systems.

6.1 Future Work

Reasoning. The focus of this paper is clearly on the application and evaluation of our inductive reasoning extension to complement the classical deductive reasoning approaches of the current Semantic Web infrastructure (see Figure 13). There exist, however, yet different types of (human) reasoning as described in [32], which were not addressed in this paper. These types are, for instance, *non-monotonic reasoning* and *temporal reasoning*. Generally speaking, in a non-monotonic reasoning system, additional/new information not considered when drawing the original conclusions can *change the reasonableness of these conclusions* [32]. In other words, the original correct conclusions are probably no longer valid and have to be revised. On the other hand, in a temporal reasoning system, the goal is to draw conclusions about the resources in the knowledgebase depending on some *notion of time*.

Without going into the details of either concepts, we think that it would make perfect sense to allow for non-monotonic and temporal reasoning facilities through the SPARQL query engine. This would allow us to derive even more additional knowledge from the asserted facts in a knowledgebase which can neither be derived by the classical deductive nor our presented inductive reasoning facilities.

Optimization. Another possible path for future work is optimization. Besides the work achieved for SPARQL basic graph pattern optimization through selectivity estimation (which we presented in [42] and [4]), we did not yet consider SPARQL-ML optimization techniques.

Generally speaking, we suggest having a closer look at virtual triple pattern optimization. Optimization in this direction will probably be twofold: first, the externally called functions need to be improved. For inductive reasoning, this implies faster algorithms to make predictions. Second, and probably more important, the query engine might need some modifications to perform query evaluation including virtual triple patterns more efficiently. This is especially important if our novel reasoning approach should be scalable and applicable to

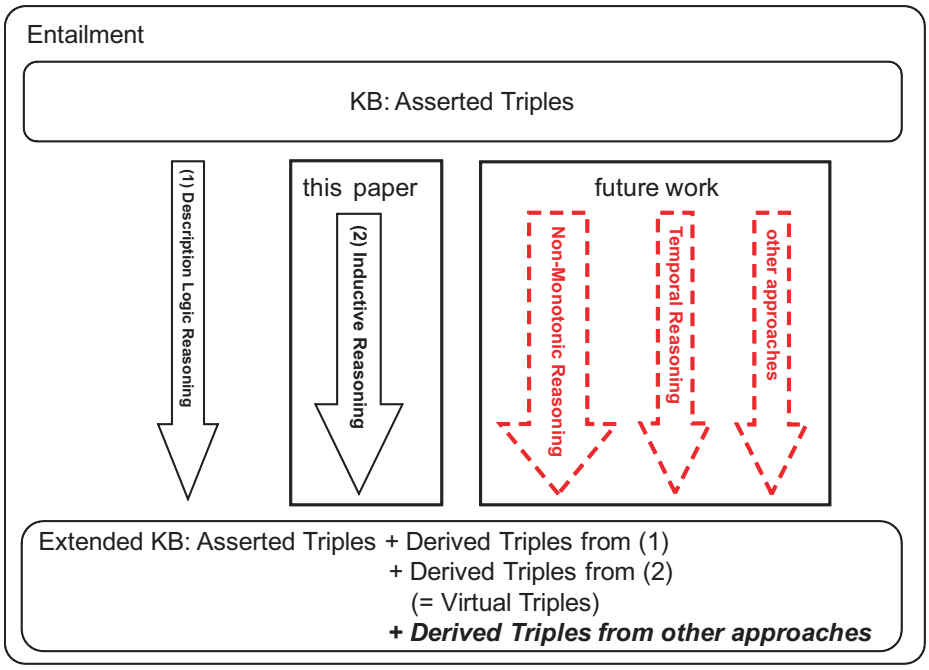

Fig. 13. Possible future reasoning extensions to the Semantic Web—*non-monotonic reasoning* and *temporal reasoning*.

datasets which are much larger than the ones used in this work (*i.e.*, if it should *scale to the Web*).

Algorithms, Datasets, and Tasks. We think that our approach's applicability to different validation tasks should be systematically investigated. An example of such a task that could substantially benefit from inductive reasoning is the classification of semantically annotated, scientific publications (as presented in the SwetoDBLP dataset).[24]

Moreover, future work should definitely evaluate the pros and cons of other relational learning methods such as the ones proposed by NetKit[25] or Alchemy.[26] This would help to underline the usefulness of this kind of learning methods for the Semantic Web.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: A Nucleus for a Web of Open Data. In: Proceedings of the 6th International Semantic

[24] `http://lsdis.cs.uga.edu/projects/semdis/swetodblp/`
[25] `http://www.research.rutgers.edu/~sofmac/NetKit.html`
[26] `http://alchemy.cs.washington.edu/`

Web Conference (ISWC). Lecture Notes in Computer Science, vol. 4825. Springer (2007)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5) (May 2001)
3. Bernstein, A., Ekanayake, J., Pinzger, M.: Improving Defect Prediction Using Temporal Features and Non-Linear Models. In: Proceedings of the 9th International Workshop on Principles of Software Evolution (IWPSE). pp. 11–18. ACM, New York, NY (2007)
4. Bernstein, A., Kiefer, C., Stocker, M.: OptARQ: A SPARQL Optimization Approach based on Triple Pattern Selectivity Estimation. Tech. Rep. IFI-2007.02, Department of Informatics, University of Zurich (2007)
5. Bizer, C., Heath, T., Ayers, D., Raimond, Y.: Interlinking Open Data on the Web. In: Proceedings of the Demonstrations Track of the 4th European Semantic Web Conference (ESWC) (2007)
6. Bloehdorn, S., Sure, Y.: Kernel Methods for Mining Instance Data in Ontologies. In: Proceedings of the 6th International Semantic Web Conference (ISWC). Lecture Notes in Computer Science, vol. 4825, pp. 58–71. Springer (2007)
7. Bloehdorn, S., Sure, Y.: Kernel Methods for Mining Instance Data in Ontologies. In: ISWC. pp. 58–71 (2007)
8. Borgida, A., Brachman, R.J., McGuinness, D.L., Resnick, L.A.: CLASSIC: A Structural Data Model for Objects. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. pp. 58–67. ACM, New York, NY (1989)
9. Broekstra, J., Kampman, A.: SeRQL: A Second Generation RDF Query Language. In: Proceedings of the SWAD-Europe Workshop on Semantic Web Storage and Retrieval (2003)
10. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 13(6), 377–387 (1970)
11. Cyganiak, R.: A relational algebra for SPARQL. Tech. Rep. HPL-2005-170, Hewlett-Packard Laboratories, Bristol (2005)
12. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM 51, 107–113 (January 2008), http://doi.acm.org/10.1145/1327452.1327492
13. Džeroski, S.: Multi-Relational Data Mining: An Introduction. ACM SIGKDD Explorations Newsletter 5(1), 1–16 (2003)
14. Edwards, P., Grimnes, G.A., Preece, A.: An Empirical Investigation of Learning from the Semantic Web. In: Proceedings of the Semantic Web Mining Workshop (SWM) co-located with 13th European Conference on Machine Learning (ECML) and the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD). pp. 71–89 (2002)
15. Fenton, N.E., Neil, M.: A Critique of Software Defect Prediction Models. IEEE Transactions on Software Engineering 25(5), 675–689 (1999)
16. Getoor, L., Licamele, L.: Link Mining for the Semantic Web. In: Dagstuhl Seminar (2005)
17. Gilardoni, L., Biasuzzi, C., Ferraro, M., Fonti, R., Slavazza, P.: Machine Learning for the Semantic Web: Putting the user into the cycle. In: Dagstuhl Seminar (2005)
18. Gruber, T.R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal Human-Computer Studies 43(5–6), 907–928 (1995)
19. Hartmann, J., Sure, Y.: A Knowledge Discovery Workbench for the Semantic Web. In: International Workshop on Mining for and from the Semantic Web (MSW). pp. 62–67 (2004)

20. Hau, J., Lee, W., Darlington, J.: A Semantic Similarity Measure for Semantic Web Services. In: Proceedings of the Workshop Towards Dynamic Business Integration co-located with the 14th International World Wide Web Conference (WWW) (2005)
21. Heß, A., Johnston, E., Kushmerick, N.: Machine Learning for Annotating Semantic Web Services. In: Semantic Web Services: Papers from the 2004 AAAI Spring Symposium Series. AAAI Press (2004)
22. Heß, A., Kushmerick, N.: Learning to Attach Semantic Metadata to Web Services. In: Proceedings of the 2nd International Semantic Web Conference (ISWC). Lecture Notes in Computer Science, vol. 2870, pp. 258–273. Springer (2003)
23. Jensen, D.: Proximity 4.3 Tutorial. Knowledge Discovery Laboratory, University of Massachusetts Amherst (2007), tutorial available at `http://kdl.cs.umass.edu/proximity/documentation.html`
24. Joachims, T.: SVM light—Support Vector Machine (2004), software available at `http://svmlight.joachims.org/`
25. Kiefer, C., Bernstein, A., Lee, H.J., Klein, M., Stocker, M.: Semantic Process Retrieval with iSPARQL. In: Proceedings of the 4th European Semantic Web Conference (ESWC). Lecture Notes in Computer Science, vol. 4519, pp. 609–623. Springer (2007)
26. Kiefer, C., Bernstein, A., Locher, A.: Adding Data Mining Support to SPARQL via Statistical Relational Learning Methods. In: Proceedings of the 5th European Semantic Web Conference (ESWC). Lecture Notes in Computer Science, Springer (February 2008), best paper award!
27. Kiefer, C., Bernstein, A., Stocker, M.: The Fundamentals of iSPARQL: A Virtual Triple Approach for Similarity-Based Semantic Web Tasks. In: Proceedings of the 6th International Semantic Web Conference. pp. 295–309 (2007)
28. Kiefer, C., Bernstein, A., Tappolet, J.: Analyzing Software with iSPARQL. In: Proceedings of the 3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE) (2007)
29. Kochut, K., Janik, M.: SPARQLeR: Extended SPARQL for Semantic Association Discovery. In: Proceedings of the 4th European Semantic Web Conference (ESWC). Lecture Notes in Computer Science, vol. 4519, pp. 145–159. Springer (2007)
30. Lam, H.Y.K., Marenco, L., Clark, T., Gao, Y., Kinoshita, J., Shepherd, G., Miller, P., Wu, E., Wong, G., Liu, N., Crasto, C., Morse, T., Stephens, S., Cheung, K.H.: AlzPharm: integration of neurodegeneration data using RDF. BMC Bioinformatics 8(3) (2007)
31. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 international conference on Management of data. pp. 135–146. SIGMOD '10, ACM, New York, NY, USA (2010), `http://doi.acm.org/10.1145/1807167.1807184`
32. Mohanan, K.P.: Types of Reasoning: Relativizing the Rational Force of Conclusions. In: Academic Knowledge and Inquiry (2008), text available at `http://courses.nus.edu.sg/course/ellkpmoh/critical/reason.pdf`
33. Neville, J., Jensen, D., Friedland, L., Hay, M.: Learning Relational Probability Trees. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 625–630. ACM, New York, NY (2003)
34. Neville, J., Jensen, D., Gallagher, B.: Simple Estimators for Relational Bayesian Classifiers. In: Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM). pp. 609–612. IEEE Computer Society, Washington, DC (2003)

35. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and Complexity of SPARQL. In: Proceedings of the 5th International Semantic Web Conference (ISWC). Lecture Notes in Computer Science, vol. 4273, pp. 30–43. Springer (2006)
36. Provost, F., Fawcett, T.: Robust Classification for Imprecise Environments. Machine Learning 42(3), 203–231 (2001)
37. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. Tech. rep., W3C Recommendation 15 January (2008), http://www.w3.org/TR/rdf-sparql-query/
38. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. 62, 107–136 (February 2006), http://portal.acm.org/citation.cfm?id=1113907.1113910
39. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall (2003)
40. Sabou, M.: Learning Web Service Ontologies: Challenges, Achievements and Opportunities. In: Dagstuhl Seminar (2005)
41. Shadbolt, N., Berners-Lee, T., Hall, W.: The Semantic Web Revisited. IEEE Intelligent Systems 21(3), 96–101 (2006)
42. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation. In: Proceedings of the 17th International World Wide Web Conference (WWW). pp. 595–604. ACM, New York, NY (2008)
43. Stutz, P., Bernstein, A., Cohen, W.W.: Signal/Collect: Graph Algorithms for the (Semantic) Web. In: Patel-Schneider, P. (ed.) ISWC 2010. vol. LNCS 6496, pp. pp. 764–780. Springer, Heidelberg (2010)
44. Valiente, G.: Algorithms on Trees and Graphs. Springer (2002)
45. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco, CA (2005)